www.sapientia.uji.es | 73



Bases de datos avanzadas

María José Aramburu Cabo
Ismael Sanz Blasco

Bases de datos avanzadas

María José Aramburu Cabo
Ismael Sanz Blasco



DEPARTAMENT D'ENGINYERIA I CIÈNCIA
DELS COMPUTADORS

■ Codi d'assignatura II52

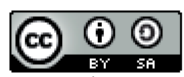
Edita: Publicacions de la Universitat Jaume I. Servei de Comunicació i Publicacions
Campus del Riu Sec. Edifici Rectorat i Serveis Centrals. 12071 Castelló de la Plana
http://www.tenda.uji.es e-mail: publicacions@uji.es

Col·lecció Sapientia, 73
www.sapientia.uji.es
Primera edició, 2013

ISBN: 978-84-695-6769-2



Publicacions de la Universitat Jaume I és una editorial membre de l'UNE, cosa que en garanteix la difusió de les obres en els àmbits nacional i internacional. www.une.es



CONTENIDOS

Capítulo 2
Sistemas de recuperación de información y documentos estructurados 47

Capítulo 3
Bases de datos distribuidas e integración de información distribuida 91

Prólogo

Esta publicación incluye los apuntes de teoría de una asignatura de cuyo nombre adopta el título de Bases de datos avanzadas. Dicha asignatura se ofrece como optativa en dos titulaciones de la Universitat Jaume I de Castellón: en cuarto curso de Ingeniería Informática (Plan 2001) y en tercer curso de Ingeniería Técnica en Informática de Gestión (Plan 2001).

En su más amplio sentido, podría decirse que las bases de datos avanzadas son todas aquellas con funcionalidades que no son propias de las bases de datos relacionales tal y como se concibieron en sus inicios por E. F. Codd. En una publicación sobre bases de datos avanzadas es posible incluir un enorme rango de modelos desarrollados con el fin de abordar aplicaciones que no pueden resolverse con las bases de datos relacionales. A continuación enumeramos un amplio grupo de tipos de bases de datos avanzadas diferentes entre sí: orientadas a objetos, objeto-relacionales, activas, multimedia, científicas, estadísticas, espaciales, temporales, multidimensionales, semiestructuradas, deductivas, difusas, con restricciones, distribuidas, federadas, móviles, multi-bd, Grid, paralelas, no-sql, etc.

Una asignatura en la que rápidamente se presenten todos estos modelos avanzados requeriría unos conocimientos previos que no son propios de estudiantes de tercer y cuarto curso, sino más bien de estudiantes de máster y doctorado. Por esta razón, cuando diseñamos los contenidos de esta asignatura, en lugar de hacer una revisión de todos estos modelos, decidimos abordar con más profundidad un reducido número de temas que consideramos fundamentales para un ingeniero en sistemas de información. En consecuencia, en esta publicación se incluyen los siguientes temas:

1. **Bases de datos orientadas a objetos**. Aunque en el mercado actual de bases de datos las orientadas a objetos no han logrado encontrar una buena posición, sí es un hecho que el modelo de bases de datos objeto-relacional ha sido paulatinamente incorporado en los sistemas de gestión de bases de datos más utilizados como Oracle o PostgreSQL. Las funcionalidades que proporcionan los sistemas objeto-relacionales han permitido desarrollar nuevas aplicaciones con tipos de datos complejos como por ejemplo las bases de datos geográficos. Los estudiantes de esta asignatura realizan prácticas en las que diseñan e implementan bases de datos objeto-relacionales con Oracle. Entendemos que una buena utilización del modelo objeto-relacional requiere adquirir previamente suficiente destreza en el manejo de los conceptos y lenguajes propios de las bases de datos orientadas a objetos.

2. **Sistemas de recuperación de información**. Ciertamente los sistemas de recuperación de información textual no pueden ser considerados sistemas de bases de datos ya que no soportan la mayoría de las funciones típicas de los sistemas de gestión de bases de datos, ni proporcionan lenguajes que permitan manipular y relacionar la información como se hace con sql. Sin embargo, hemos encontrado suficientes razones para incluir este tema en

una asignatura de bases de datos avanzadas. La razón principal es que estos sistemas ofrecen una serie de tecnologías que hoy en día se requieren en muchos sistemas de información que además de datos estructurados deben gestionar documentos. De hecho, desde hace ya varios años, los principales sistemas de gestión de bases de datos han sido extendidos con módulos para la indexación y recuperación de textos planos y documentos estructurados, principalmente en formato XML. Los estudiantes de esta asignatura realizan prácticas con Oracle para almacenar y recuperar textos planos y documentos XML. A través de este capítulo esperamos que sean capaces de comprender el funcionamiento de los sistemas de recuperación de información, y también de utilizar estos conocimientos para explotarlos mejor tanto a la hora de recuperar información como de desarrollar nuevos sistemas de información textual.

3. **Bases de datos distribuidas**. En el último capítulo de esta publicación hemos querido proporcionar a los estudiantes algunos conocimientos básicos para iniciarlos en la compleja tarea de desarrollar sistemas de información que integren datos provenientes de varias fuentes. Internet ha facilitado enormemente el intercambio de datos entre fuentes separadas entre sí, así como el diseño de sistemas de bases de datos en los que las aplicaciones y los ficheros de información se almacenan distribuidos en sitios remotos. No obstante, desde el punto de vista de su diseño y eficiencia, estas bases de datos tienen una serie de peculiaridades que dificultan su utilización. El objetivo de este capítulo es que los estudiantes comprendan los requerimientos propios de un sistema de información distribuido y se inicien en los métodos disponibles para su diseño y desarrollo.

Esperamos que esta publicación sirva de ayuda tanto para estudiantes de la titulación Ingeniería Informática como para profesionales de la informática que requieran actualizar sus conocimientos. Sin embargo, probablemente antes de que pase mucho tiempo sus contenidos deberán ser revisados, ya que las bases de datos siguen avanzando rápidamente, al igual que el resto de áreas de la ingeniería informática.

CAPÍTULO 1

Bases de datos orientadas a objetos

1. Introducción

Las bases de datos relacionales son, hoy en día, las más utilizadas en la mayoría de los ámbitos de aplicación. La estructura de datos básica que ofrece, la tabla relacional, es apropiada para muchas aplicaciones habituales. Sin embargo, existen casos de uso en los que presenta serios inconvenientes prácticos, generalmente si se requiere gestionar datos muy complejos o no convencionales (imágenes, documentos...), para los que las estructuras relacionales resultan muy complejas e ineficientes. Algunos ejemplos de gran importancia práctica son las bases de datos multimedia, las bases de datos científicos y los sistemas de apoyo al diseño industrial (CAD/CAM).

Las bases de datos orientadas a objetos intentan dar respuesta a estos problemas, incorporando las siguientes características:

- Adoptan como modelo de datos el de los lenguajes orientados a objetos, permitiendo así el uso de estructuras de datos tan complejas como sea necesario, y eliminando en gran medida las barreras entre el desarrollo de aplicaciones y la gestión de datos.

- Permiten la extensibilidad con nuevos tipos de datos complejos, permitiendo incorporar operaciones arbitrarias sobre ellos.

Estas características han motivado el desarrollo de numerosos sistemas orientados a objetos, que se han establecido en nichos en los que los casos de uso mencionados son importantes. Además, la actual emergencia de aplicaciones web que requieren estructuras de datos muy flexibles está motivando un gran interés en sistemas capaces de soportarlas, como es el caso de las bases de datos orientadas a objetos.

En este capítulo estudiaremos las principales características del modelo datos orientado a objetos, así como de los sistemas basados en él. También introduciremos conceptos sobre el desarrollo y uso de estos sistemas: diseño lógico, físico y lenguajes de consulta.

1.1. Objetivos de aprendizaje

Los objetivos de aprendizaje de este capítulo se enumeran a continuación:

a) Explicar los principales casos de uso de las bases de datos orientadas a objetos.
b) Diferenciar las bases de datos orientadas a objetos, las objeto-relacionales y las relacionales.
c) Explicar los principales conceptos del modelo de datos orientado a objetos.
d) Explicar el concepto de persistencia de objetos y sus implementaciones típicas en la práctica.

e) Explicar las principales características de los sistemas de gestión de bases de datos orientados a objetos.
f) Describir las principales primitivas de diseño lógico orientado a objetos.
g) Diseñar una base de datos orientada a objetos a partir de un diagrama entidad/relación.
h) Realizar consultas simples sobre bases de datos orientadas a objetos usando el lenguaje OQL.
i) Describir los principales métodos de diseño físico para bases de datos orientado a objetos.

2. Evolución histórica de las bases de datos

Los modelos de datos en que se basan los sistemas de gestión de bases de datos han ido evolucionando con el tiempo como se muestra en la figura 1.1. Esta evolución se ha visto determinada por las necesidades de cada momento. Por ejemplo, hasta bien entrada la década de 1980 prácticamente las únicas aplicaciones en que se usaban bases de datos eran de tipo comercial: contabilidad, gestión de pedidos de clientes, etc. Por ello, los modelos de datos predominantes hasta ese momento (jerárquicos, de red y relacional) estaban orientados al manejo de los tipos de datos que requieren esas aplicaciones, que son muy sencillos: números (especialmente para almacenar cantidades) y textos cortos (para describir asientos contables, o nombres y direcciones de clientes). Estos modelos y en especial el relacional, han tenido una gran aceptación para desarrollar la tecnología de bases de datos necesaria en las aplicaciones tradicionales de negocios y gestión.

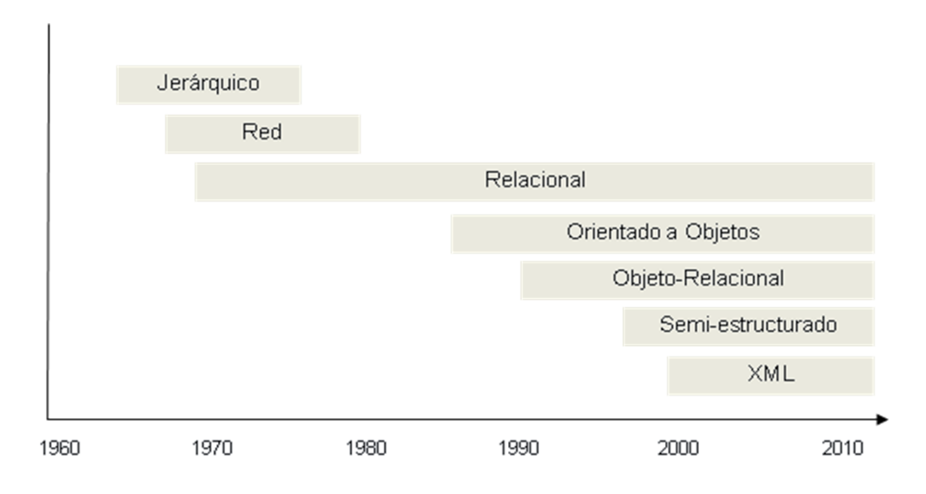


Figura 1.1. Evolución de los modelos de bases de datos.

Sin embargo, a mediados de los años ochenta del pasado siglo, con la explosión del uso de los microordenadores a nivel empresarial y la aparición de los tipos de datos multimedia, se expandieron enormemente los tipos de aplicaciones en los que era necesario utilizar bases de datos, y se hicieron evidentes las limitaciones en los modelos de datos tradicionales: no eran capaces de tratar los tipos de datos complejos que eran necesarios, por ejemplo, en aplicaciones como el diseño y la

fabricación asistidas por ordenador (CAD/CAM, CIM), las bases de datos gráficas y de imágenes, las bases de documentos y multimedia, y los sistemas de información geográfica. Estas nuevas aplicaciones tienen requerimientos y características diferentes a los de las aplicaciones de negocios: estructuras más complejas para los datos, transacciones de mayor duración, nuevos tipos de datos para almacenar imágenes o grandes bloques de texto, y la necesidad de definir operaciones específicas para cada aplicación. En aquellos momentos las bases de datos relacionales no disponían de mecanismos adecuados para implementar las nuevas aplicaciones.

Las bases de datos orientadas a objetos (BDOO) se propusieron con el objetivo de satisfacer las necesidades de las aplicaciones anteriores, al proporcionar un modelo de datos mucho más rico y extensible, y así complementar (aunque no sustituir) a las bases de datos relacionales. Además, el modelo de datos proporcionado por las BDOO es equivalente al de los lenguajes de programación orientados a objetos, como C++ o Java. Esto tiene la gran ventaja de que, al compartir el modelo de datos, se pueden integrar las BDOO con el software usado para desarrollar aplicaciones, de manera directa y casi transparente. En cambio, en las aplicaciones basadas en bases de datos relacionales es necesario usar dos lenguajes muy diferentes (un lenguaje de programación para la aplicación, y SQL para el acceso a la base de datos), lo que implica realizar costosas conversiones entre los formatos de datos usados en cada uno de los dos lenguajes.

La necesidad de disponer de nuevas características para el modelado de datos complejos ha sido reconocida por los principales vendedores de sistemas de gestión de bases de datos relacionales. Así las últimas versiones de estos sistemas han ido incorporando muchas de las funcionalidades que inicialmente se propusieron para las BDOO dando lugar a las bases de datos objeto-relacionales (BDOR). Las últimas versiones del estándar SQL, a partir de SQL99, también incluyen muchas de estas funcionalidades. Entre ellas se incluyen la posibilidad de almacenar en tablas relacionales grandes objetos binarios (denominados «BLOBS», por *Binary Large OBjects*), como imágenes, sonido o video; las herramientas de gestión de documentos (índices y operadores específicos para buscar texto, soporte de XML); y otras extensiones para soportar datos geográficos. Las últimas versiones del estándar SQL, a partir de SQL99, también dan soporte a muchas de estas características.

Por lo tanto, podemos hablar de varios niveles de soporte de la tecnología orientada a objetos en bases de datos:

- A pequeña escala se encuentran las librerías que permiten el almacenamiento persistente de objetos. Estas están disponibles para cualquier lenguaje de programación orientado a objetos, pero en muchos casos no incorporan características avanzadas, como soporte transaccional o multiusuario.
- Después están las bases de datos objeto-relacionales, para aplicaciones que requieren usar algunos tipos de datos complejos en un entorno esencialmente relacional, y que se basan en extensiones orientadas a objetos de SQL.
- Finalmente, las bases de datos orientadas a objetos puras proporcionan una gestión de bases de datos orientadas a objetos a todos los niveles, desde la definición de datos al lenguaje de consulta.

A este respecto, y de manera análoga a las «12 reglas de Codd» que establecen los requisitos que debe cumplir una base de datos relacional, se han descrito una serie de reglas para poder caracterizar hasta qué punto una base de datos soporta características orientadas a objetos. Estas reglas se describirán en la sección 4.2. de este capítulo.

3. Conceptos del modelo de datos orientado a objetos

En esta sección se definen los diferentes conceptos que componen el modelo de datos de las bases de datos orientadas a objetos: identidad de los objetos, constructores de tipos y objetos, referencias entre objetos, estado de los objetos, comportamiento de los objetos, y finalmente clasificación e instanciación de objetos. La sección termina con una figura que resume y relaciona todos estos conceptos.

3.1. Objetos

En las bases de datos orientadas a objetos todos los elementos que se manejan son objetos. Cada objeto se corresponde con una entidad de la realidad, y define sus propiedades y comportamiento. Formalmente, tal y como se representa en la figura 1.2, un *objeto* es una representación abstracta de una entidad del mundo real que tiene una *identidad única* dentro de la base de datos, unas *propiedades* incorporadas en sí mismo, y un *comportamiento* que le proporciona la capacidad de interaccionar con otros objetos y consigo mismo. Los objetos que comparten propiedades y comportamiento forman clases, de las que trataremos más adelante.

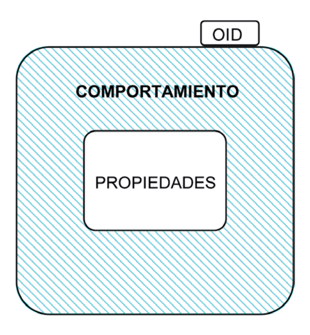


Figura 1.2. Componentes de un objeto

Los objetos, al igual que las filas de las bases de datos relacionales, expresan las propiedades de los elementos de la realidad. Sin embargo, los objetos, al contrario que las tuplas, tienen identidad y son elementos activos, de tal forma que poseen un comportamiento y pueden interaccionar entre sí.

3.2. Identidad

La parte más importante de un objeto es su identidad única. La identidad de cada objeto se representa por medio de un OID (*Object Identifier*), el cual es único para ese objeto. De esta forma es imposible que dos objetos puedan compartir el mismo OID. El OID es asignado por el sistema en el momento en que el objeto es creado y no puede ser alterado bajo ninguna circunstancia. El OID de un objeto no es visible para el usuario externo, sino que el sistema lo utiliza internamente para identificar al objeto, y para referenciarlo desde otros objetos.

La diferencia entre un OID y una clave primaria está en que la clave primaria se basa en los valores dados por los usuarios para ciertos atributos, los cuales pueden ser alterados en cualquier momento. Sin embargo, el OID es asignado por el sistema, debe ser independiente del valor de cualquier atributo, y no puede ser alterado. El OID solo puede desaparecer cuando su objeto lo haga, y en ese caso nunca puede volver a ser utilizado. Por último, el OID tampoco puede referirse a ninguna dirección de memoria. Así se consigue que la manera en que se representan los objetos y sus relaciones sea independiente del formato de almacenamiento físico de los datos.

Dos objetos con el mismo OID se consideran *idénticos*: se trata a todos los efectos del mismo objeto. Es importante remarcar que dos objetos que no sean idénticos pueden ser *iguales* si sus valores también lo son. Se trata de dos conceptos de igualdad diferentes, que tanto los lenguajes como las bases de datos orientadas a objetos diferencian claramente. Por ejemplo, en el lenguaje Java se usa el operador == para comprobar si dos objetos son idénticos, y el método `equals()` para comprobar si sus valores son iguales. Por ejemplo:

```
public class IdentidadOIgualdad {
  public static void main(String[] args) {
    String a = new String("Hola");
    String b = new String("Hola");

    System.out.println(a == b);
    System.out.println(a.equals(b));
  }
}
```

Dado que los dos objetos `a` y `b` no son idénticos (operador ==) pero sí de igual valor (método `equals()`), como resultado de ejecutar el código anterior se muestra como resultado:

```
false
true
```

3.3. Propiedades de los objetos

Cada objeto de una base de datos tiene asignado un valor que expresa sus propiedades y que puede ser actualizado a lo largo del tiempo. El valor de un objeto se ajusta a una estructura de datos que puede ser tan compleja como sea necesario. Esta estructura de datos se construye a partir de unos tipos de datos básicos por medio de unos constructores de tipos de datos y de tipos de objetos, que se pueden combinar arbitrariamente.

Entre los tipos de datos básicos que proporcione un sistema de gestión de bases de datos orientadas a objetos (SGBDOO) también pueden aparecer tipos de datos multimedia y texto. En este caso, el sistema también debe proveer las operaciones necesarias para manipular los valores multimedia y textuales. Asimismo, los SGBDOO más avanzados permiten al usuario definir sus propios tipos de datos básicos con sus propias operaciones. Esto hace al sistema muy flexible y capaz de soportar los datos de cualquier aplicación.

3.3.1. Constructores de objetos

Los *constructores* se dividen entre constructores de *átomos* (elementos de los dominios básicos: enteros, reales, cadenas, etc.), y de *colecciones*, que suelen ser tuplas ([…]), conjuntos ({…}) y listas (//…//). Las tuplas tienen varios componentes llamados atributos con un nombre y un tipo. Los conjuntos son un número de 0 o más elementos del mismo tipo y que no se repiten. Las listas son como los conjuntos pero conservando el orden de inserción. Los BDOO, como los lenguajes de programación OO, pueden proporcionar otros tipos de colecciones.

Los valores de los objetos se construyen a partir de los átomos (valores reales, enteros, cadenas, etc.) y también a partir del conjunto de OID de todos los objetos existentes en la base de datos. A la hora de construir los valores, los constructores pueden ser anidados dando lugar a valores con estructuras complejas. Por ejemplo, los siguientes objetos representarían en una base de datos a dos objetos: el primero a un empleado y el segundo al departamento que dirige.

```
o1 = (OID = oid_e1,
      valor = ['José García',
               '99.999.999-X',
               [1980, 12, 10],
               V,
               oid_d1
              ] )

o2 = (OID= oid_d1,
      valor= ['Informática',
               17,
               [oid_e1, [1999, 10, 17] ],
               {'Castellón', 'Valencia', 'Alicante'},
               {oid_e1, oid_e13, oid_e11, oid_e17, oid_e5, oid_e3, oid_e2}
             ] )
```

Los lenguajes de definición de datos orientados a objetos permiten definir los tipos de los datos y los tipos de objetos necesarios para construir los objetos de una aplicación concreta. En los ejemplos de este capítulo, utilizaremos un lenguaje de definición de datos simplificado, basado en el lenguaje O_2C del SGBDOO O_2. Las palabras reservadas tuple, set y list permiten definir los tipos tupla, conjunto y lista respectivamente, y para los tipos atómicos asumiremos que están disponibles los tipos de datos básicos más habituales (integer, string, real...). En este lenguaje, los tipos de los dos objetos anteriores se definen de la siguiente manera.

```
define data type Fecha
tuple [  año:     integer,
         mes:     integer,
         dia:     integer]

define object type Empleado
      tuple [   nombre:     string,
                dni:        string,
                fecha_nac:  Fecha,
                sexo:       char,
                depto:      Departamento]

define object type Departamento
      tuple [   nombred:    string,
                numerod:    integer,
                dirigido:   tuple [ gerente: Empleado,
                                    fecha_inic: Fecha],
                sedes:      set(string),
                empleados:  set(Empleado)]
```

Como puede verse en el ejemplo, hemos definido un tipo de datos (data type) Fecha para simplificar la definición los tipos de objetos (object type) Empleado y Departamento. Los tipos de datos se utilizan para simplificar la definición de tipos de objetos, y representan estructuras de datos que se reutilizan en la definición de las propiedades de distintos tipos de objetos (como fechas o direcciones), y no van a tener existencia independiente: en ningún caso va haber objetos «Fecha» con identidad propia. Es muy importante hacer notar esta distinción entre «object type» para definir objetos, y «data type» que se utiliza exclusivamente para facilitar la definición de las propiedades de los tipos de objetos.

3.3.2. Referencias entre objetos

Todo atributo de un tipo tiene un dominio asignado, que puede ser un tipo de datos o un tipo de objetos previamente definidos. Como puede verse en el ejemplo anterior, cuando el valor de un atributo es de un tipo de objetos (p.ej. depto), en lugar de asignar al atributo todo el valor, se le asigna el OID del objeto. En ningún caso se copia el valor del objeto destino (salvo que así se decidiera explícitamente), sino que se usa el OID.

Así, los atributos cuyo dominio sea un tipo de objetos en realidad se implementan como *referencias* usando el OID, y representan relaciones entre los objetos. Una relación binaria entre dos objetos se puede representar en una sola dirección o en

ambas direcciones (*referencia inversa*). Esta última representación hace más fácil recorrer la relación en ambas direcciones, y será la que nosotros usaremos exclusivamente.

En el ejemplo anterior, el tipo Empleado tiene un atributo depto que hace referencia a un objeto de tipo Departamento. Esta relación también se ha representado en la dirección inversa ya que el tipo Departamento tiene definida una componente empleados como un conjunto de objetos Empleado. Para asegurar la integridad de las referencias cruzadas el usuario puede utilizar una cláusula inverse de la misma manera que en el ejemplo siguiente. También hay que fijarse en que la relación gerente solo se ha representado en una dirección, por lo que no lleva cláusula inverse asociada. La definición completa de estos dos tipos se presenta a continuación.

```
define object type Empleado
        tuple[      nombre:      string,
                    dni:         string,
                    fecha_nac:   Fecha,
                    sexo:        char,
                    depto:       Departamento inverse Departamento:empleados]

define object type Departamento
        tuple[      nombred:     string,
                    numerod:     integer,
                    dirigido:    tuple[ gerente: Empleado,
                                        fecha_inic: Fecha],
                    sedes:       set(string),
                    empleados:   set(Empleado) inverse Empleado:depto]
```

3.3.3. Estado de los objetos

El *estado* de un objeto es el valor que tiene asignado dicho objeto en un momento determinado. Esto significa que cuando las propiedades de un objeto cambian, su valor debe ser actualizado en la base de datos, haciendo pasar el objeto a un nuevo estado.

Como se ha indicado anteriormente, es posible que en algún momento dos objetos no idénticos (con distinto OID) sean iguales (tengan el mismo valor). Es posible definir distintos conceptos de igualdad entre valores, como veremos con un ejemplo. Consideremos los siguientes objetos:

o_1 = (OID= *oid_1*, valor= ['Hola'])
o_2 = (OID= *oid_2*, valor= ['Hola'])
o_3 = (OID= *oid_3*, valor= [oid_1])
o_4 = (OID= *oid_4*, valor= [oid_2])

Los objetos o_1 y o_2 son claramente iguales, ya que tienen el mismo valor. Sin embargo, la relación entre o_3 y o_4 es ambigua. Por un lado, sus valores son diferentes, pero por otro lado, si seguimos las referencias, comprobamos que ambas apuntan a objetos que tienen el mismo valor. Esto nos lleva a las siguientes definiciones:

- La *igualdad superficial*, que se comprueba comparando los valores del estado, sin seguir referencias.

- La *igualdad profunda*, que se comprueba comparando los valores del estado, y (recursivamente) los valores de los objetos referenciados.

Con estas definiciones:

- Los objetos o_1 y o_2 son iguales tanto superficial como profundamente.
- Los objetos o_3 y o_4 no son iguales superficialmente, pero sí profundamente.

Esta distinción también se suele tener en cuenta al copiar los valores de los objetos, y en muchos casos se proporcionan operaciones distintas para copiar los valores tanto superficialmente como profundamente. Por ejemplo, en el lenguaje Python existen las funciones `copy.copy()` y `copy.deepcopy()`, que realizan la copia superficial y profunda de un objeto respectivamente.

3.4. Comportamiento de los objetos

El comportamiento de los objetos se representa por medio de operaciones que se ejecutan sobre ellos. Toda operación que deba realizarse sobre un objeto tiene que estar previamente implementada. Las operaciones se usan para crear y eliminar objetos, para cambiar los valores de los atributos de los objetos, para conocer su valor, o para realizar operaciones mucho más complejas que pueden involucrar a muchos objetos. En general las operaciones representan las acciones del mundo real, es decir, representan el comportamiento de los objetos.

Cada operación tiene una signatura y un método. La *signatura* es la parte pública de la operación, mientras que el *método* es un procedimiento implementado en algún lenguaje de programación y que permanece oculto para los usuarios. Los métodos que se asocian a cierto objeto pueden acceder directamente a sus atributos y hacer estos contenidos visibles al exterior, pudiendo realizar alguna computación previa sobre ellos. Por ejemplo, se puede escribir un método que lea la fecha de nacimiento de un empleado y proporcione como resultado la edad de dicho empleado.

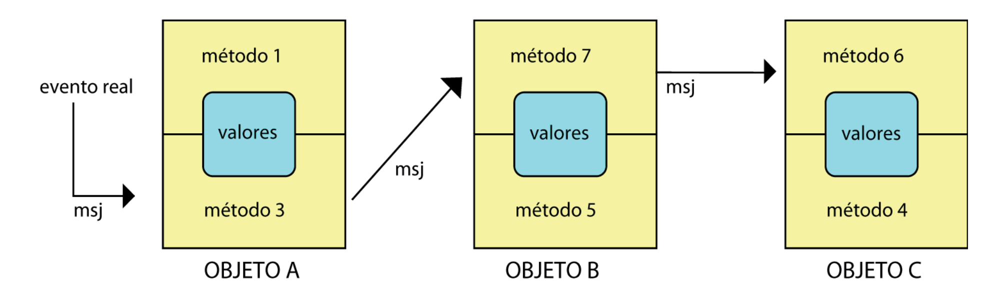


Figura 1.3. Comunicación entre objetos

La ejecución de una operación se conceptualiza como el envío de un *mensaje* a un objeto: un mensaje se envía a un determinado objeto, y contiene el nombre de la operación y los parámetros necesarios para que el método asociado se ejecute.

El método se ejecuta sobre el objeto que recibe el mensaje. Como ilustra la figura 1.3, estos métodos pueden enviar sus propios mensajes a otros objetos y utilizar el resultado devuelto en su computación sobre el objeto inicial.

Es importante saber que la estructura interna de los objetos (o sea, su valor) puede ocultarse, lo que impide que pueda ser accedida directamente si no es por medio de un método. Esta habilidad de esconder los detalles internos de los objetos se denomina *encapsulación*, y es una de las ventajas más importantes del modelo orientado a objetos, ya que facilita la manipulación de los objetos desde las aplicaciones, al tiempo que asegura su integridad.

3.5. Clasificación e instanciación de objetos

Las bases de datos orientadas a objetos clasifican los objetos de acuerdo a sus similitudes. Una *clase* es un conjunto de objetos que comparten unas propiedades (tipo de objetos) y un comportamiento (operaciones). La *instanciación* es el proceso inverso al de la clasificación y consiste en generar los objetos de una clase. Veamos el siguiente ejemplo de definición de las clases de una base de datos:

```
define object type Empleado extent Los_empleados
       type
         tuple[  nombre:          string,
                 dni:             string,
                 fecha_nac:       Fecha,
                 sexo:            char,
                 depto:           Departamento inverse Departamento:empleados]
       operations
         crear_nuevo_empleado:Empleado,
         destruir_emp():boolean,
         edad():integer;


define object type Departamento extent Los_departamentos
       type
         tuple[  nombred:         string,
                 numerod:         integer,
                 dirigido:        tuple[gerente: Empleado, fecha_inic: Fecha],
                 sedes:           set(string),
                 empleados:       set(Empleado) inverse Empleado:depto]
       operations
         crear_nuevo_depto:Departamento,
         destruir_depto():boolean,
         numero_de_emps():integer,
         añadir_emp(e:Empleado):boolean,
         quitar_emp(e:Empleado):boolean;
```

Cada objeto de la base de datos es instancia de una clase. El conjunto de instancias de una clase se denomina su *extensión*, a la que se da un nombre explícito mediante la cláusula extent. Para crear y eliminar las instancias, toda clase debe disponer de un método de creación (constructor) y otro de eliminación de instancias (destructor).

El conjunto de operaciones de una clase constituye su aspecto público y se denomina *interfaz* o *protocolo de acceso*. Los cuerpos de los métodos se implementan

aparte utilizando alguno de los lenguajes de programación disponibles en el SGB-DOO que se esté utilizando. Normalmente el SGBDOO está acoplado a un lenguaje de programación orientado a objetos para implementar las operaciones de los objetos y las aplicaciones de las bases de datos.

Para terminar esta sección, la figura 1.4 muestra de manera gráfica los distintos conceptos del modelo de datos orientado a objetos, y las relaciones entre ellos.

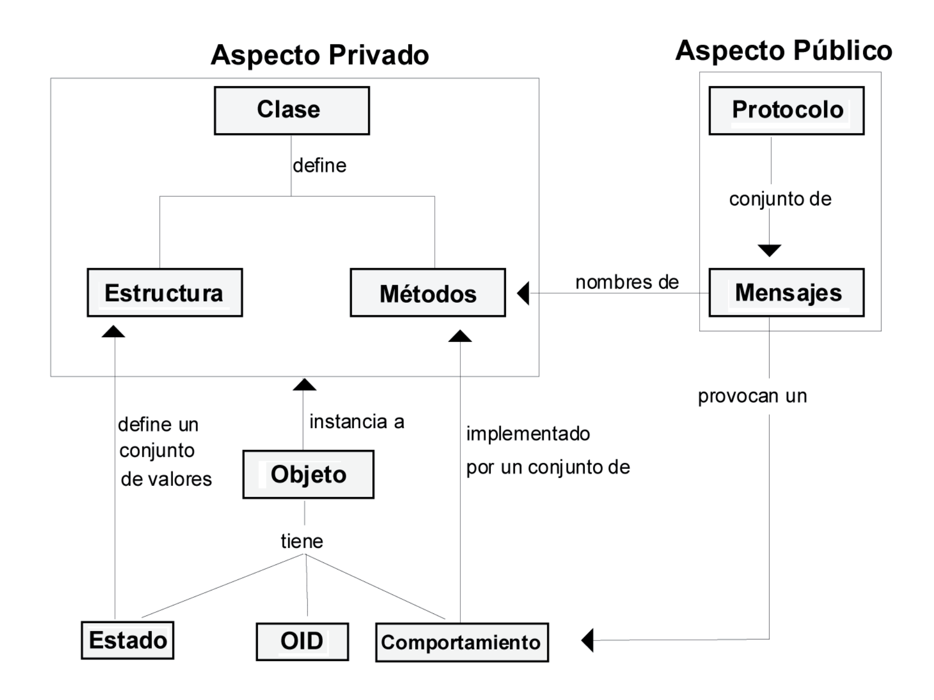


Figura 1.4. Resumen de conceptos básicos del modelo orientado a objetos

4. Sistemas de gestión de bases de datos orientados a objetos

Como se ha comentado anteriormente, el hecho de que las bases de datos orientadas a objetos y los lenguajes de programación orientados a objetos compartan el mismo modelo de datos permite desarrollar aplicaciones sobre BDOO de manera casi transparente, en contraste con las laboriosas transformaciones de datos que son necesarias en las aplicaciones desarrolladas sobre bases de datos relacionales. Sin embargo, hay que tener en cuenta los siguientes factores:

- Desde el lenguaje de programación orientado a objetos es necesario indicar qué objetos se guardarán en la base de datos (serán *persistentes*) y cuáles no; esto se trata en la sección 4.1 de este capítulo.

- Según los requisitos de la aplicación a desarrollar, se deberá utilizar un SGB-DOO con determinadas características. En la sección 4.2 de este capítulo se describen estas características, siguiendo el *Manifiesto BDOO*, análogo a las 12 reglas de Codd del modelo relacional.

4.1. Persistencia de objetos

A la hora de implementar aplicaciones sobre BDOO, una decisión muy importante es qué objetos deberán permanecer en la base de datos una vez que el programa haya terminado, y cuáles no. Los *objetos transitorios* desaparecen una vez que el programa que los creó termina de ejecutarse, mientras que los *objetos persistentes* permanecen almacenados en la base de datos.

Para hacer un objeto persistente hay tres alternativas. La primera es marcarlo explícitamente como persistente. En el siguiente ejemplo en Java se utiliza la BDOO *db4o* (*db4objects*) para crear un objeto de tipo `Empleado`, que a continuación se marca como persistente usando el método `store` para tal efecto. El objeto `contenedor` encapsula el almacenamiento de objetos.

```
Empleado empleado = new Empleado("Joan", "Cardona", "Vives");
contenedor.store(empleado);
```

A partir de este momento, todos los cambios que se produzcan en este objeto se guardarán automáticamente. Al marcar un objeto como persistente existe la posibilidad de asignarle un nombre único dentro de la base de datos, a través del cual el SGBDOO permite a los usuarios y a los programas acceder a él. Los objetos con nombre único suelen ser pocos y sirven como puntos de acceso a la base de datos.

La segunda alternativa para hacer un objeto persistente es hacerlo alcanzable a partir de otro objeto ya marcado como persistente. Se dice que un objeto A es alcanzable desde otro B, si hay alguna cadena de referencias que llevan desde el objeto B hasta el objeto A. Esta cadena de referencias puede ser muy larga y pasar por muchos objetos. De esta manera, cuando se hace un objeto persistente, todos los objetos alcanzables a partir de él se hacen implícitamente persistentes.

Siguiendo con el ejemplo anterior en *db4o*, supongamos que el tipo de objetos `Empleado` contiene una referencia a objetos de una clase `Direccion`. Por ejemplo:

```
class Empleado {
      ⁝
      private Direccion direccion;
      ⁝
      public void setDireccion(Direccion nueva_dir) {
               dirección = nueva_dir;
      }
⁝
}
```

Podemos crear un objeto de tipo `Empleado` y asignarle una dirección de esta manera:

```
Direccion direccion = new Direccion("C/Mayor, 5", "Castellón")
Empleado empleado = new Empleado("Joan", "Cardona", "Vives");
empleado.setDireccion(direccion);
contenedor.store(empleado);
```

No ha sido necesario marcar explícitamente como persistente la instancia de `Direccion` que hemos creado, ya que es alcanzable desde el objeto `empleado` y, por tanto, se guardará y recuperará automáticamente. Es decir, el método `store` hace persistente al objeto `empleado` y a todos los objetos alcanzables a partir de él.

En la práctica, los sistemas suelen permitir mantener un control muy preciso sobre qué objetos se harán persistentes por alcanzabilidad, ya que hay muchos casos en que no es deseable seguir ciegamente todas las cadenas de referencias (estructuras de datos circulares, valores temporales que no es necesario guardar, etc.). Por ejemplo, para que el código anterior funcione correctamente deberíamos configurar explícitamente *db4o* para que la clase Empleado use persistencia por alcanzabilidad sin restricciones, ya que por defecto no es así. Los detalles concretos son diferentes en cada sistema.

La última manera de hacer objetos persistentes es añadiéndolos a una *colección persistente* de su clase. Sobre una clase se pueden definir varias colecciones de objetos persistentes, y para aquellas clases que vayan a tener instancias almacenadas en la base de datos es imprescindible tener definida al menos una. Por ejemplo, en nuestro lenguaje de definición de datos se define por medio de la cláusula extent una colección persistente para cada clase. De esta manera, para hacer un objeto persistente, solo hay que añadirlo a una de las colecciones persistentes asociadas a su clase.

Siguiendo con el ejemplo, en *db4o* se encuentran versiones persistentes de las colecciones estándar; por ejemplo `com.db4o.collections.ActivatableLinkedList` es la versión persistente de la colección estándar `java.util.LinkedList`, y tiene la misma interfaz:

```
ActivatableLinkedList<Empleado> listaDeEmpleados;
listaDeEmpleados = new ActivatableLinkedList<Empleado>();
Empleado empleado = new Empleado("Joan", "Cardona", "Vives");
listaDeEmpleados.add(empleado)
```

4.2. Características de los SGBDOO

Edgar F. Codd, el creador del modelo relacional, definió en 1985 una serie de 12 reglas que permiten establecer las características propias de un sistema de gestión de bases de datos relacional. El conjunto de reglas equivalente para los sistemas orientados a objetos se conoce como el *Manifiesto de las BDOO*, y fue definido en 1992. Consta de las 13 reglas que se detallan en la tabla 1.1 (obligatorias para considerar que un SGBD es OO) y de cinco opciones de implementación.

Al contrario que las reglas, las cinco opciones de implementación no son obligatorias. Son las siguientes:

1. Herencia múltiple
2. Verificación e inferencia de tipos
3. Distribución de los datos
4. Transacciones
5. Versionado de objetos

Es importante hacer notar que las reglas 1 a 8 son características generales de los sistemas orientados a objetos, incluyendo los lenguajes de programación, y que por otro lado, las reglas 9 a 13 son características generales de las bases de datos, incluyendo las no orientadas a objetos, como las relacionales.

Como en el caso de las 12 reglas de Codd, ninguna base de datos comercial cumple todas las reglas del *Manifiesto de las BDOO*, por lo que puede decirse que estas reglas describen un sistema idealizado no disponible en la realidad. Sin embargo, en la práctica se trata de una lista muy útil para analizar los SGBDOO y determinar si se ajustan a los requisitos de una aplicación dada.

Regla	Explicación
1. Objetos complejos	Un SGBDOO podrá gestionar objetos con estructuras de datos arbitrarias (Ver sección 3.3 Propiedades de los objetos).
2. Identidad de objetos	En una BDOO los objetos tendrán una identidad, que será diferente de su valor (Ver sección 3.2 Identidad).
3 Encapsulación	Un SGBDOO permitirá ocultar los detalles de implementación de los objetos, haciendo que un objeto solo sea accesible mediante su interfaz público (Ver sección 3.4 Comportamiento de los objetos: mensajes y métodos).
4. Tipos y clases	Un SGBDOO permitirá definir tipos de datos y tipos de objetos (Ver sección 3.5 Clasificación e instanciación de objetos).
5. Jerarquías de clases	Un SGBDOO permitirá organizar los objetos en clases, y permitirá definir nuevas clases especializando o generalizando otras ya existentes (Ver sección 5.2 Generalización, especialización y herencia).
6. Sobrecarga y polimorfismo	Distintas clases podrán proporcionar distintos métodos para una misma operación; el sistema determinará dinámicamente qué método se debe ejecutar (Ver sección 5.3 Polimorfismo de métodos).
7. Lenguaje computacionalmente completo	Un SGBDOO proporcionará un lenguaje de programación computacionalmente completo, que es aquel que puede expresar cualquier algoritmo. SQL, por ejemplo, no es completo.
8. Extensibilidad	Cada SGBDOO tiene un conjunto de tipos predefinidos por el sistema. Este conjunto puede ser extendido. Los tipos definidos por el sistema y los tipos definidos por el usuario deben ser tratados del mismo modo.
9. Persistencia	En una SGBDOO, los objetos serán persistentes de la manera más transparente posible (Ver sección 4.1 Persistencia).
10. Gestión de memoria secundaria	Todo SGBDOO ha de ofrecer funciones de gestión de almacenamiento eficiente. Esto ha de incluir el uso de índices, agrupación de datos, el almacenamiento en memoria intermedia de datos, selección de la ruta de acceso y optimización de consultas. Todas estas características deben ser invisibles para el programador de la aplicación.
11. Concurrencia	Un SGBDOO debe ofrecer mecanismos para sincronizar el acceso de más de un usuario al mismo tiempo.
12. Recuperación	Un SGBDOO debe mantener el nivel de servicios en caso de fallos de software o hardware.
13. Consultas ad hoc con un lenguaje declarativo	Un SGBDOO proporcionará un lenguaje de consulta específico para objetos (Ver sección 6. Consultas en BDOO).

Tabla 1.1. Características de los SGBDOO

5. Diseño lógico de bases de datos orientadas a objetos

Para diseñar bases de datos orientadas a objetos hay que identificar las clases de objetos que se necesitan, así como el tipo y las operaciones de cada clase. Para definir el tipo de los objetos de una clase, primero hay que identificar qué relaciones se dan entre las clases. En esta sección, previamente a definir un procedimiento de diseño, vamos a analizar los distintos tipos de relaciones que se pueden dar entre las clases de una BDOO.

5.1. Agregación y asociación

Los objetos agregados son aquellos que se componen de otros objetos. La *agregación* es un tipo de relación que permite relacionar objetos agregados con sus objetos componentes (relación todo/parte). Representar al objeto agregado puede ser especialmente útil cuando el propio objeto agregado se debe relacionar con otros objetos, o viceversa, cuando alguno de los objetos componentes se debe relacionar con otro objeto. Como se ilustra en la figura 1.5, en los esquemas conceptuales de la base de datos, la relación entre un objeto y sus componentes siempre la vamos a denotar con se_compone_de y dibujaremos flechas que vayan desde el objeto agregado hacia los componentes. En una *jerarquía de agregación* un objeto agregado se puede componer de varios objetos componentes, pero un objeto componente solo puede formar parte de un objeto agregado.

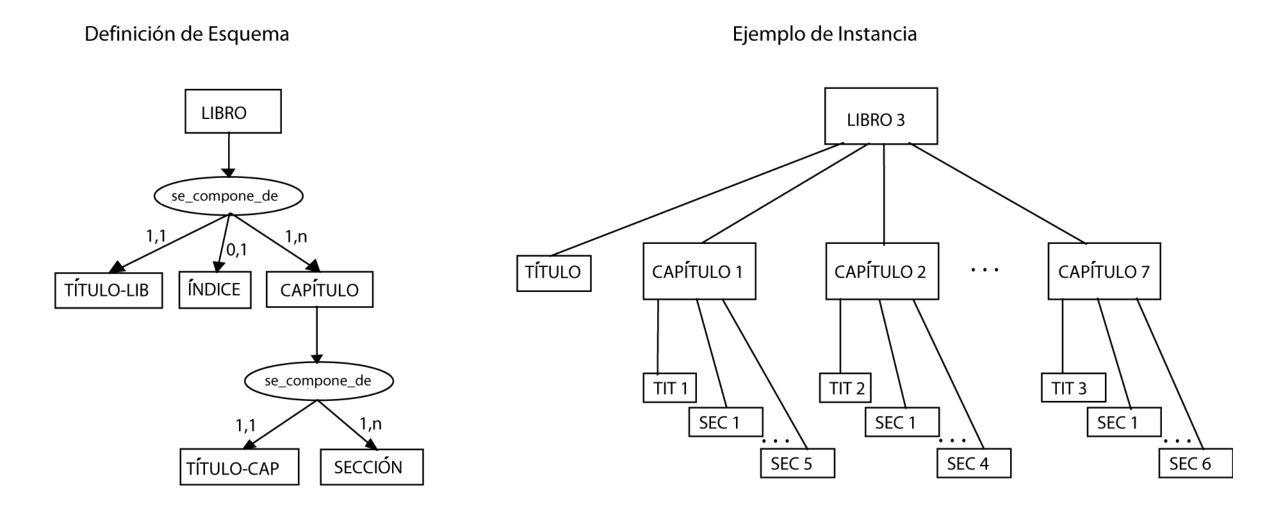


Figura 1.5. Relaciones de agregación

La relación de *asociación* sirve para relacionar objetos de varias clases independientes. En el esquema conceptual de la base de datos a este tipo de relaciones se las denomina con un nombre y una cardinalidad que reflejen su significado. En el esquema conceptual de la figura 1.6 aparecen cuatro relaciones de asociación diferentes.

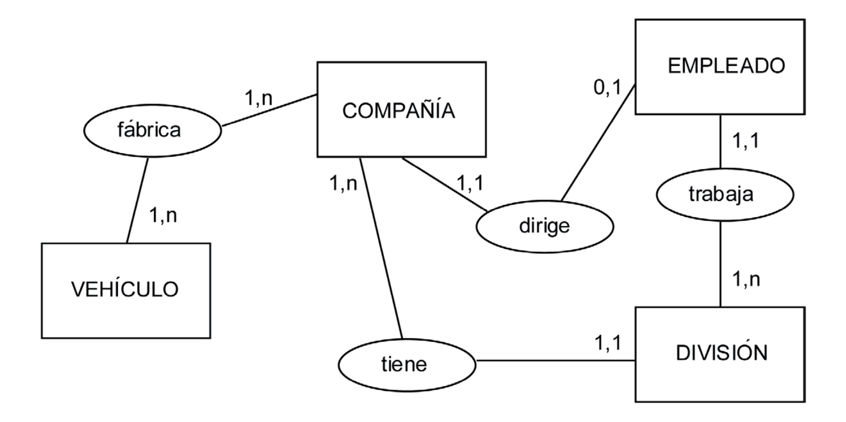


Figura 1.6. Relaciones de asociación

En cierta manera la agregación es un tipo especial de asociación. La diferencia principal entre ellas es que cuando se elimina un objeto que participa en una relación de asociación, los otros objetos relacionados siguen existiendo. Sin embargo, en el caso de la agregación, la inserción/eliminación del objeto agregado equivale a insertar/eliminar sus objetos componentes. Similarmente, un objeto que forma parte de otro no puede ser insertado ni eliminado de la base de datos individualmente. Lógicamente, un objeto componente nada más puede formar parte de un objeto compuesto.

A la hora de realizar el diseño lógico de una BDOO, cuando se definen los tipos de los objetos de una clase, las relaciones de asociación y de agregación se representan por medio de referencias en una o ambas direcciones, y de acuerdo a su carnalidad asociada. En algunos SGBDOO es posible diferenciar en el esquema las relaciones de agregación de las de asociación. En nuestro caso, las representamos de la misma forma.

5.2. Generalización, especialización y herencia

Como ya hemos visto, una clase sirve para organizar un conjunto de objetos similares. Sin embargo, en muchas ocasiones los objetos de una clase pueden reorganizarse, formando varios grupos que también pueden tener interés para la base de datos. Por ejemplo, en la figura 1.7 vemos que los objetos de la clase empleado pueden clasificarse en secretarios, ingenieros, gerentes y técnicos. Se dice que cada una de estas agrupaciones es una *subclase* de la clase empleado, y empleado es una *superclase* de las otras cuatro clases. Nótese que todo ingeniero es un empleado pero no viceversa. Es decir, todas las instancias de una subclase son también instancias de la superclase, pero no a la inversa.

Este tipo de relación se denomina *generalización*, o *especialización* a la inversa, y en el esquema conceptual se representa con una relación denominada es_un y unas flechas que van desde las subclases hacia las superclase. La generalización tiene dos propiedades de las bases de datos orientadas a objetos:

- La propiedad de *sustituibilidad* dice que toda instancia de una clase puede ser utilizada cuando se espera una instancia de alguna de sus superclases.
- La propiedad de *extensión* dice que las instancias de una clase incluyen las instancias de sus subclases. En otras palabras, la extensión de una clase incluye a las extensiones de sus subclases.

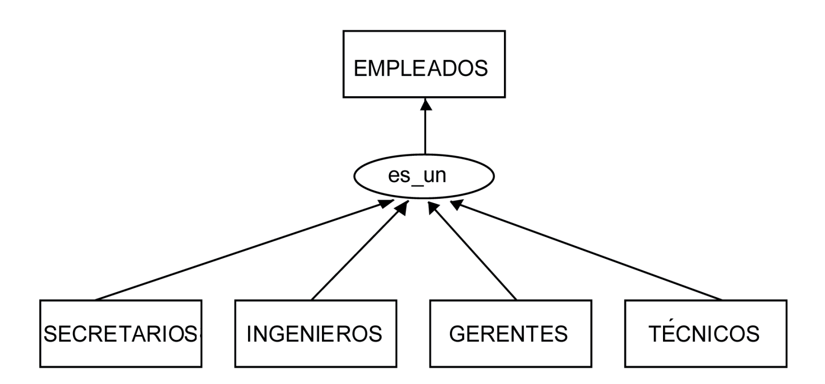


Figura 1.7. Relaciones de especialización

Como ejemplo, observemos el siguiente código Java. Asumiendo que tenemos definida la jerarquía de clases que se muestra en la figura anterior:

```
public class Generalizacion {
   public static void main(String[] args) {
     Secretario sec = new Secretario("Francesc", "Tárrega");
     Gerente ger = new Gerente("Maximilià", "Alloza");
     System.out.println(sec instanceof Empleado);
     System.out.println(ger instanceof Empleado);
   }
}
```

El resultado de ejecutar este código es el siguiente:

```
true
true
```

mostrando que los secretarios y los gerentes están incluidos en la extensión de los empleados.

Cuando el mecanismo de especialización se aplica sucesivamente en varios niveles de clases se crea una *jerarquía de especialización* (ver figura 1.8). Una importante característica de las jerarquías de especialización es el de la *herencia* de tipos y operaciones desde las superclases hacia las subclases. Es decir, todas las subclases de una clase heredan automáticamente su tipo y operaciones, siendo esta una propiedad que se propaga a lo largo de toda la jerarquía. Por esta razón, a la hora de hacer el diseño lógico, la definición del tipo y de las operaciones de una subclase incluye de manera implícita a todas las definiciones hechas para la superclase más aquellas nuevas definiciones (nuevos atributos y nuevas operaciones) explícitas que quieran hacerse.

Existen dos tipos de herencia: simple y múltiple. La *herencia simple* ocurre cuando cada clase en una jerarquía tiene una sola superclase inmediata. Si se envía un mensaje a un objeto de una clase en una jerarquía con herencia simple, el método asociado se busca primero en la clase y luego en su superclase inmediata, y así sucesivamente hacia arriba en la jerarquía. La *herencia múltiple* ocurre cuando una subclase tiene varias superclases inmediatas. En este caso se heredan los tipos y las operaciones de todas las superclases, por lo que pueden surgir colisiones con los nombres de los atributos y de los mensajes. En ese caso se produciría un error. En las relaciones de la figura 1.8 se producen ambos tipos de herencia.

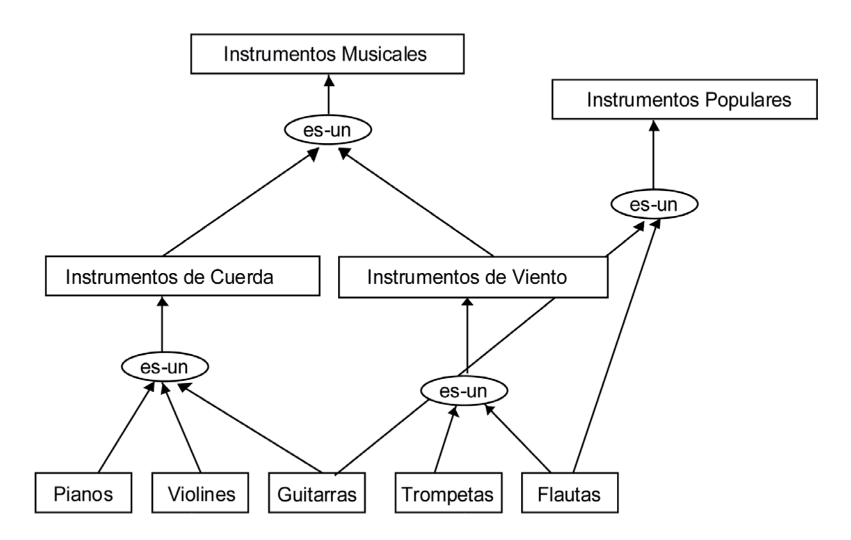


Figura 1.8. Jerarquía de clases con herencia múltiple

5.3. Polimorfismo de métodos

Otra característica de las BDOO es que hacen posible el *polimorfismo* de los métodos. Este concepto permite asignar a una misma operación dos o más implementaciones diferentes, con el propósito de que en cada caso se ejecute una de ellas, dependiendo de la clase de los objetos a los que se aplique. Aunque no siempre es un requerimiento necesario, lo normal es que las distintas clases formen parte de una jerarquía de especialización a través de la cual se hereda el método que va a ser redefinido.

Por ejemplo, supongamos que se define una clase para objetos geométricos que cuenta con una operación que calcula su área, y con tres especializaciones para los rectángulos, los triángulos y los círculos.

```
define object type Objeto_Geometrico
type
        forma: string
operations
        area(): real;


define object type Rectangulo inherit Objeto_Geometrico
type
```

```
        tuple [   ancho: real,
                  alto: real]
  operations
        es_cuadrado?(): bool;


define object type Triangulo inherit Objeto_Geometrico
  type
        tuple [   lado_1: real,
                  lado_2: real,
                  angulo: real]
  operations
        es_equilatero?(): bool;

define object type Circulo inherit Objeto_Geometrico
  type
        radio:    real;
```

La operación area() se declara para todos los objetos de la clase Objeto_Geometrico, aunque su implementación varía dependiendo de la subclase del objeto. Una posibilidad es tener una implementación general para calcular el área de cualquier objeto geométrico, y luego implementar otros métodos específicos para calcular las áreas de ciertos objetos geométricos como los triángulos o los círculos. A la hora de ejecutar el método sobre cierto objeto, el SGBDOO debe seleccionar la implementación apropiada en base a la clase del objeto geométrico de que se trate. La búsqueda se hace de abajo a arriba, es decir el sistema buscará la definición del método en la clase del objeto o en la primera superclase suya que tenga dicha definición.

Por ejemplo, en el siguiente código Java se asume que se ha creado la función area() en todas las subclases de Objeto_Geometrico.

```
public class Polimorfismo {
   public static void main(String[] args) {
       // Creamos una colección de Objeto_Geometrico
       List<Objeto_Geometrico> objetos = new List<Objeto_Geometrico>();
       // Añadimos instancias de objetos geométricos
       objetos.add(new Rectangulo(2, 2));
       objetos.add(new Triangulo(3, 2, Math.PI/4));
       objetos.add(new Circulo(1.5));

       // Calculamos el área de todos los objetos
       // Se llamará en cada caso al método correspondiente a la subclase
       for (Objeto_Geometrico o : objetos) {
           System.out.println(o.area());
         }
   }
}
```

En este ejemplo se aprovechan las propiedades de sustituibilidad y polimorfismo. La sustituibilidad permite crear una colección de objetos de la superclase Objeto_Geometrico, y añadir objetos de las subclases. Además, se puede llamar al método polimórfico area() sin preocuparnos de cuál es la subclase a que pertenece cada objeto, lo que simplifica muchísimo la programación.

5.4. Diseño lógico de una base de datos OO

Para implementar una BDOO a partir de un esquema conceptual lo primero que hay que hacer es definir los tipos de datos, los tipos de objetos y las operaciones de las clases en el lenguaje de definición de datos de nuestro SGBD. Posteriormente, se añaden los métodos que se implementan en el lenguaje de programación disponible. Vamos a partir de un esquema conceptual que represente tanto las relaciones de asociación, como las de agregación y especialización, sin embargo los métodos tendrán que ser especificados por separado. A grandes rasgos la transformación puede hacerse como sigue:

Paso 1: Crear una clase de objetos para cada entidad del esquema. El tipo de objetos de la clase deberá agrupar todos los atributos de la entidad por medio de un constructor de tupla. Los atributos multivaluados se declaran a través de un constructor de tipo conjunto, o lista si están ordenados. Los atributos complejos se declaran por medio de un constructor de tupla. Todos los tipos y, sobre todo, aquellos tipos de datos que se utilicen en varias clases se pueden definir como tipos de datos aparte. Todas las clases deben tener una extensión (cláusula extent) definida.

Paso 2: Aquellas clases que son subclases de otras deberán tener una cláusula inherit para indicarlo. Cada subclase hereda de forma automática el tipo y las operaciones de su superclase inmediata. Solo hay que especificar los atributos y operaciones específicos de la subclase.

Paso 3: Para cada relación de asociación y agregación en las que participe una clase, añadir atributos para referenciar las clases que participen en la relación. Las referencias pueden definirse en una dirección o en ambas, aunque suele ser más eficiente definirlas en ambas. Nosotros añadiremos atributos con referencias a las dos clases relacionadas, excepto en las relaciones de cardinalidad (0,1) que dependerá de cada caso a analizar. Los atributos serán monovaluados para las relaciones de cardinalidad (1,1), y multivaluados (atributos de tipo conjunto o lista) para los de cardinalidad (1,n). Si una relación se representa con referencias en ambas direcciones se debe declarar que cada referencia es la inversa de la otra (cláusula inverse). Si hay alguna relación con atributos, hay que usar un constructor de tupla para representarlo de la siguiente manera tuple[referencia con cláusula inverse, atributos de la relación]. Esta tupla se incluye en el atributo de referencia.

Paso 4: Incluir métodos apropiados para cada clase. Estos no estarán disponibles en el esquema conceptual y se deberán agregar al diseño de la base de datos según se necesiten. Al final, toda clase debe incluir un método constructor y otro destructor para sus instancias.

5.5. Ejemplo de diseño de una base de datos orientada a objetos

A continuación se define un ejemplo de una base de datos para una universidad que se ajusta al esquema conceptual de la figura 1.9. Para simplificar el esquema no hemos puesto los atributos de las entidades, también vamos a suponer que los métodos para crear y destruir los objetos de cada clase son definidos por el sistema.

Primero se definen dos tipos de datos.

```
define data type Telefonos: set(integer);

define data type Fecha:tuple[año:integer, mes:integer, dia:integer];
```

A continuación, se definen los tipos de objetos que se corresponden con las entidades del diagrama conceptual.

```
define object type Persona extent Personas
        type
                tuple[  dni: string,
                        nombre: tuple [  nombrepila:string,
                                         apellido_p:string,
                                         apellido_s:string],
                        direccion:tuple [ numero:integer,
                                         calle:string,
                                         ciudad:string],
                        fechanac:Fecha
                        sexo:char]
        operations
                edad():integer;

define object type Profesor inherit Persona extent Profesores
        type
                tuple [ salario: real,
                        rango: string,
                        telefono: Telefonos,
                        pertenece_a: Departamento inverse Departamento:miembros,
                        tutoriza:set(Estudiante_Investigacion)inverseEstudiante_investigacion:tutor,
                        imparte: set(Asignatura) inverse Asignatura:prof]
        operations
                promover_profesor:boolean,
                dar_aumento(porcentaje:real):boolean;
```

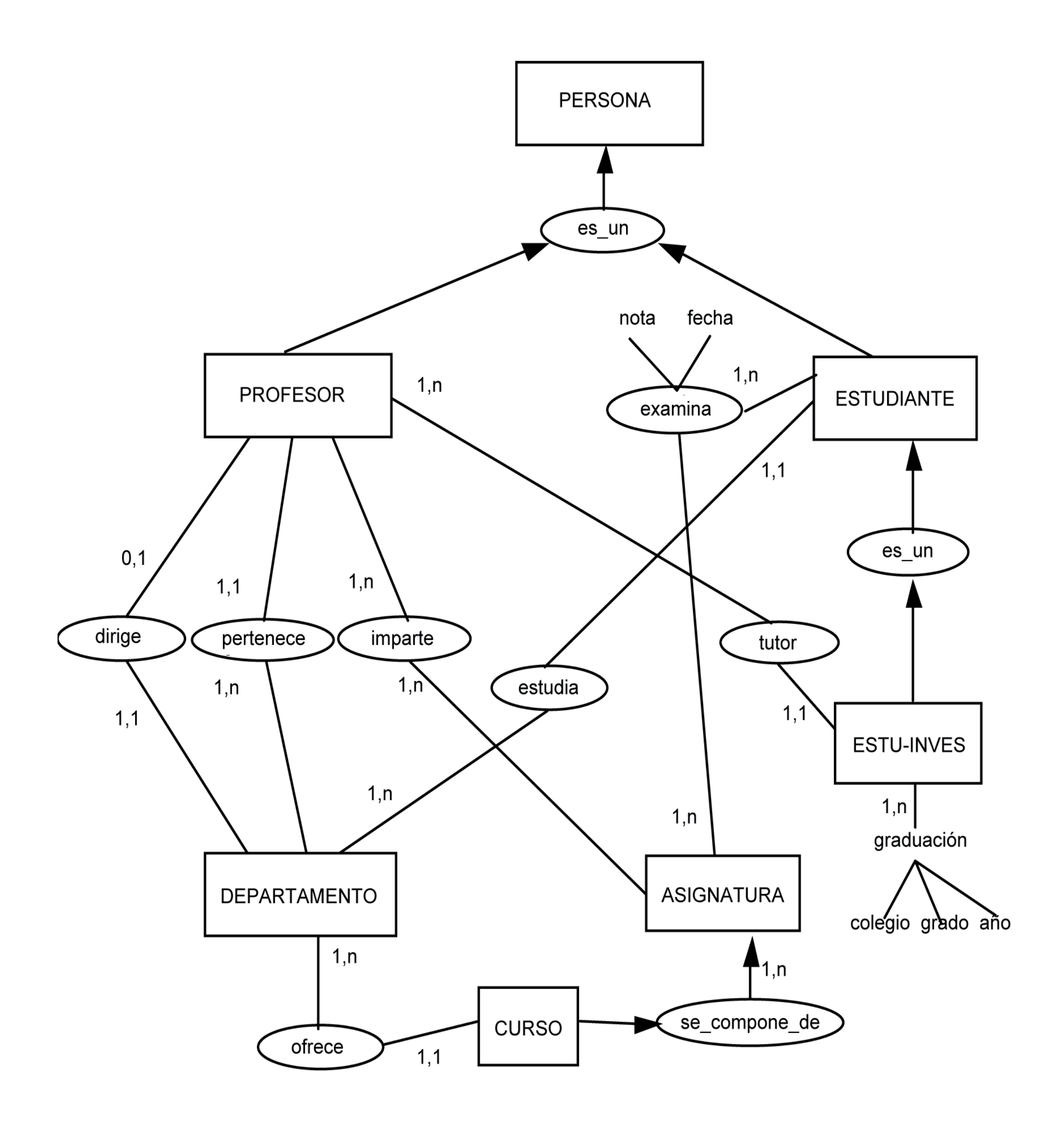


Figura 1.9. Esquema conceptual de la BD de una universidad

```
define object type Estudiante inherit Persona extent Estudiantes
        type
          tuple [  grupo: string,
                   estudia: Departamento inverse Departamento:alumnos,
                   examinado_de: set(tuple [  nota:real,
                                              fecha:Fecha,
                                              asignatura: Asignatura inverse
                                                      Asignatura:examinados.estudiante]) ]
        operations
          nota_media():real,
          cambiar_grupo:boolean,
          cambiar_carrera(nueva_carrera:Departamento):boolean;

define object type Estudiante_investigacion inherit Estudiante extent Estudiantes_inv
        type
          tuple [  telefono:Telefonos,
                   graduación: list(tuple [    colegio:string,
                                              grado:string,
                                              año:integer]),
                   tutor: Profesor inverse Profesor:tutoriza];
```

```
define object type Departamento extent Departamentos
        type
          tuple [ nombre: string,
                  oficinas: set(string),
                  telefonos: Telefonos,
                  miembros: set(Profesor) inverse Profesor:pertenece_a,
                  alumnos: set(Estudiante) inverse Estudiante:estudia,
                  director: Profesor,
                  cursos: set(Curso) inverse Curso:ofrecido_por]
        operations
          añadir_profesor(p:Profesor),
          añadir_a_carrera(e:Estudiante),
          quitar_de_carrera(e:Estudiante):boolean;

define object type Curso extent Cursos
        type
          tuple [ nombre: string,
                  codigo: string,
                  descripcion: string,
                  asignaturas: set(Asignatura) inverse Asignatura:curso,
                  ofrecido_por: Departamento inverse Departamento:cursos]
        operations
          añadir_asignatura(a:Asignatura);


define object type Asignatura extent Asignaturas
        type
          tuple [ cod_asig: integer,
                  año: integer,
                  examinados:set(tuple [    nota:real,
                                            fecha:Fecha,
                                            estudiante:Estudiante inverse
                                                      Estudiante:examinado_de.asignatura])
                  curso: Curso inverse Curso:asignaturas,
                  profesores: set(Profesor) inverse Profesor:imparte ]
        operations
          cambiar_notas(e:Estudiante, n:integer);
```

Vamos a terminar el ejemplo de base de datos con una propuesta de implementación para el método edad() de la clase Persona.

```
method body edad:integer in class Persona
{
  int a; Date d;
  d=today();
  a=d->año - self->fechanac->año;
  if (d->mes < self->fechanac->mes) ||
     (( d->mes == self->fechanac->mes) && (d->dia < self->fechanac->dia)) --a
}
```

6. Consultas en bases de datos orientadas a objetos

A la hora de realizar consultas a las BDOO se han propuesto diferentes lenguajes más o menos expresivos y fáciles de utilizar. Todavía no existe ningún lenguaje estándar oficial, por lo que aquí nos vamos a centrar en un lenguaje que se aproxima mucho a uno que podría ser el más aceptado. Se trata del lenguaje OQL (*Object Query Language*) que ha sido diseñado por un consorcio de compañías de *software* denominado ODMG (*Object Data Management Group*). Hoy día, el estándar está mantenido por la organización OMG, dedicada a la estandarización de las tecnologías orientadas a objetos.

El lenguaje OQL cuenta con una estructura select...from...where... similar a la de SQL. Por ejemplo, para recuperar el nombre de los departamentos presididos por la profesora llamada 'Ana' puede escribirse la siguiente consulta:

```
select d.nombre
from Departamentos d
where d.presidente.nombre.nombrepila = 'Ana'
```

6.1. Puntos de acceso y variables iterador

La cláusula from de cada consulta debe incluir un *punto de acceso* a la base de datos que puede ser cualquier objeto persistente con nombre específico. Normalmente, el punto de acceso será el nombre de la colección persistente asociada a una de las clases involucradas en la consulta. Hay que considerar que el nombre de una colección (cláusula extent) se corresponde con el nombre de un objeto persistente cuyo tipo es un conjunto de objetos de una clase. Por esta razón al escoger el nombre de una colección como punto de acceso a la base de datos estamos haciendo referencia a un conjunto de objetos persistentes que pueden ser recuperados si cumplen las condiciones de la consulta.

Lo más recomendable es acceder a la base de datos por la colección de aquella clase sobre la que las condiciones de la consulta (claúsula where) establezcan condiciones más restrictivas. De esta manera el resto de condiciones de la consulta se tienen que evaluar para un menor número de objetos, y la consulta se ejecuta mucho más eficientemente.

Dentro de claúsula from de una consulta siempre que se haga referencia a un conjunto (o lista) de objetos es necesario definir una *variable iterador* que toma un valor por cada objeto de la colección (en el ejemplo anterior d es la variable iterador). Esta variable se puede utilizar para poner condiciones sobre los objetos y para acceder a otros objetos de la base de datos como veremos en la siguiente sección.

El resultado de una consulta siempre es un conjunto de objetos que, a menos que se utilice la cláusula distinct, podrá contener elementos repetidos. El tipo de estos elementos se deduce de lo que se especifique en la cláusula select. En general, el

resultado de una consulta puede ser de cualquiera de los tipos que podamos definir con los constructores de tipos disponibles. En el ejemplo anterior, dado que el atributo nombre de la clase departamento es de tipo string, el resultado de la consulta será de tipo set(string).

6.2. Caminos de acceso

Los *caminos de acceso* o *trayectorias* además de permitir referirse a los componentes de los objetos complejos, pueden ser utilizados para navegar entre los objetos a través de sus relaciones de agregación y asociación. De esta manera, si obj es un objeto que tiene el atributo att, entonces obj.att denota el valor del atributo para ese objeto concreto. Alternativamente, si meth es un método de la clase a la que obj pertenece, entonces obj.meth() denota el valor resultante de aplicar al objeto obj el método dado. Esta manera de especificar los caminos de acceso se denomina *notación de punto*.

Por ejemplo, al escribir e.estudia.nombre donde e es un objeto de la clase Estudiante, primero se localiza al departamento relacionado con el estudiante e a través de su atributo estudia y luego se recupera el atributo nombre de dicho departamento. Otros ejemplos de caminos de acceso sobre las clases Persona y Estudiante de la base de datos anterior son las siguientes:

```
p.nombre.nombrepila np
p.direccion.calle c
e.nombre.nombrepila ne
e.estudia.telefonos t
```

Las variables iterador que aparecen después de los caminos de acceso denotan los valores que se almacenan en los atributos donde estas terminan. Sin embargo, hay que tener en cuenta que cuando una referencia es multivaluada no se puede usar la notación de punto directamente. En este caso es preciso intercalar una nueva variable iterador, como se ha hecho en los siguientes caminos de acceso que empiezan en la clase Profesor:

```
p.pertenece_a.miembros m, m.nombre.nombrepila np
p.imparte a, a.examinados e, e.estudiante.estudia.nombre nd
```

En este último ejemplo, la variable iterador e no toma como valores objetos, sino los valores complejos que se almacenan en el atributo examinados. Dado que las referencias se implementan almacenando el OID de los objetos relacionados, se puede decir que los caminos de acceso sustituyen a los *joins* de las bases de datos relacionales. Esto quiere decir que al consultar una BDOO no se deben especificar relaciones de *join* entre sus clases, sino que se deben utilizar variables iterador y caminos de acceso como las que se muestran en los ejemplos anteriores.

6.3. Lenguaje de consulta

La estructura básica de OQL es select...from...where..., similar a la de SQL, pero con la diferencia de que es posible usar caminos de acceso dentro de la consulta para referirse a los componentes de los objetos complejos y para pasar de unos objetos a otros relacionados. Otras diferencias, motivadas por los distintos modelos de datos que subyacen a ambos lenguajes, se ilustrarán en los ejemplos que se muestran más adelante.

En una consulta, la cláusula select define los datos que se desean conocer y el tipo de los datos del resultado de la consulta. La cláusula from especifica las extensiones de objetos por los que se accede a la base de datos y proporciona las variables iterador que toman como valores los objetos de dichas colecciones. Estas variables pueden utilizarse en el resto de cláusulas de la consulta. Por último, la cláusula where especifica las condiciones para seleccionar los objetos individuales que intervienen en el resultado de la consulta. A continuación vamos a ver algunos ejemplos de consultas sobre la base de datos anterior:

a) Para conocer los distintos salarios que tienen los profesores de la base de datos podemos escribir la siguiente sentencia. Se obtendrá un conjunto de números reales diferentes.

```
select distinct p.salario
from Profesores p
```

b) Para conocer los distintos salarios que hay para cada sexo se puede definir la siguiente consulta:

```
select distinct tuple[salario:p.salario, sexo:p.sexo]
from Profesores p
```

En este caso se obtiene como resultado un conjunto de tuplas diferentes entre sí, donde cada tupla se compone del salario y sexo de un profesor de la extensión Profesores. Es decir, el resultado de la consulta será un valor del tipo set(tuple[salario:real, sexo:char]).

c) Las colecciones de la cláusula from también pueden especificarse por medio de caminos de acceso. Por ejemplo, para conocer el nombre de las asignaturas del año 2010 de que cada estudiante se ha examinado se puede escribir una de las dos consultas siguientes:

```
select tuple[ nombre:e.nombre,
              cod_asig: a.asignatura.cod_asig]
from Estudiantes e, e.examinado_de a
where a.asignatura.año = 2010


select tuple[ nombre:e.estudiante.nombre,
              cod_asig: a.cod_asig]
from Asignaturas a, a.examinados e
where a.año = 2010
```

La diferencia entre las dos consultas es su punto de acceso a la base de datos, siendo más rápida la segunda versión ya que accede a la base de datos por la misma clase por la que se ponen las condiciones del where, además de que en la base de datos hay muchos más estudiantes que asignaturas.

d) Veamos esto por medio de otro ejemplo. Supongamos que se desean conocer los números del DNI de aquellos estudiantes matriculados en alguna asignatura impartida por un profesor de Sagunto. Para evaluar esta consulta podríamos elegir entre las dos opciones siguientes:

```
select distinct e.dni
from Estudiantes e, e.examinado_de a, a.asignatura.profesor p
where p.direccion.ciudad = 'Sagunto'
```

```
select distinct e.estudiante.dni
from Profesores p, p.imparte a, a.examinados e
where p.direccion.ciudad = 'Sagunto'
```

La diferencia entre cada una de las opciones es su tiempo de ejecución. Accediendo a la base de datos por la extensión Estudiantes, los dos caminos de acceso del from se tendrán que evaluar para cada estudiante de la base de datos, ya que sobre los estudiantes no se pone ninguna restricción. Sin embargo, accediendo por la extensión de Profesores, los dos caminos de acceso se tienen que evaluar para un menor número de objetos, ya que lo normal es que en la base de datos haya muchos menos profesores de Sagunto que estudiantes. Por lo tanto, la segunda opción se evaluará mucho más rápidamente que la primera.

e) En la cláusula where se puede especificar cualquier condición sobre los objetos denotados por las variables de la cláusula from. Por ejemplo, si solamente se está interesado en aquellos estudiantes que cursen su carrera en el departamento de informática y que estén examinados en más de 5 ocasiones tenemos que escribir la siguiente consulta que devolverá un conjunto de objetos de la clase estudiante.

```
select e
from Departamentos d, d.alumnos e
where d.nombre='Informática'
and count(e.examinado_de) > 5
```

Este ejemplo muestra cómo se puede utilizar la función count para saber cuántos valores se almacenan en el atributo examinado_de. Esta misma función se puede utilizar para saber cuántos OID se almacenan en un atributo que representa una referencia 1:N.

f) En la especificación de condiciones también se pueden invocar métodos para conocer las propiedades de los objetos a recuperar. Por ejemplo, la siguiente consulta recupera los estudiantes cuya media de notas hasta el momento supere el 7.

```
select e
from Estudiantes e
where e.nota_media( ) > 7.0
```

g) También se pueden definir condiciones entre los atributos siempre que sean del mismo tipo. Por ejemplo, podemos buscar parejas estudiante/profesor que cumplan años el mismo día:

```
select tuple[  estudiante: e.nombre,
               profesor: p.nombre,
               cumpleaños: tuple[mes:p.fechanac.mes, dia:p.fechanac.dia] ]
from Estudiantes e, Profesores p
where e.fechanac.mes = p.fechanac.mes
and e.fechanac.dia = p.fechanac.dia
```

h) Para obtener los resultados ordenados existe la cláusula order by. Por ejemplo, se pueden recuperar los profesores ordenados por su sueldo y por su primer apellido.

```
select p
from Profesores p
order by p.salario, p.nombre.apellido_p
```

i) También se pueden agrupar los resultados de una consulta según algún criterio de los objetos recuperados. Por ejemplo, la siguiente consulta cuenta los profesores que pertenecen al departamento de informática agrupados según la ciudad donde viven.

```
select tuple[ciudad:p.direcccion.ciudad, num_prof:count(partition)]
from Departamentos d, d.miembros p
where d.nombre = 'Informática'
group by p.direccion.ciudad
```

Como siempre accedemos a la base de datos por la clase sobre la que se ponen las condiciones del where. La palabra clave partition sirve para denotar cada uno de los grupos que se forman al procesar la consulta. En este caso la función count cuenta cuantos elementos hay en cada partición, o sea en cada grupo formado por el criterio p.direccion.ciudad.

j) Algunos atributos almacenan conjuntos o listas de valores u objetos diferentes entre sí. Se puede comprobar si todos los valores almacenados en un atributo de un objeto cumplen una condición, o si solamente alguno de ellos lo hace.

```
select c
from Cursos c
where for all a in c.asignaturas: a.año = 2010
```

```
select e
from Estudiantes e
where exists a in e.examinado_de: a.nota = 10.0
```

La primera consulta recupera los cursos con todas sus asignaturas en el año 2010, y la segunda, los estudiantes con alguna matrícula de honor entre sus notas. Esta última consulta sería equivalente a:

```
select e
from Estudiantes e, e.examinado_de a
where a.nota = 10.0
```

Para la primera consulta no hay sentencia equivalente.

k) También están disponibles las funciones de agregación similares a las de SQL: avg, count, sum, min y max. Por ejemplo, para conocer la media de edad de todos los estudiantes se puede ejecutar la sentencia siguiente:

```
select avg(e.edad()) from Estudiantes e
```

l) El resultado de una consulta puede ser asignado a un identificador para poder ser utilizado en otras consultas. Por ejemplo, se puede crear una colección de objetos de la clase Profesor que incluye a los mejor pagados utilizando la siguiente consulta:

```
define collection Mejor_pagados as
select p from Profesores p where p.salario> 60000
```

m) Ahora es posible conocer el rango de salarios de los profesores mejor pagados de la siguiente forma:

```
select tuple [maximo:max(b.salario), minimo:min(b.salario)]
from Mejor _pagados b
```

n) Si se deseara conocer el nombre de los profesores con mayor salario tendríamos que utilizar una subconsulta de la siguiente forma:

```
select p.nombre
from Profesores p
where p.salario = (select max(p.salario) from Profesores p)
```

o) Otro ejemplo de consulta es la siguiente, que devuelve un conjunto de tuplas con los detalles de las notas de cada estudiante del grupo IG2:

```
select tuple[  nombrep:e.nombre.nombrepila,
               apellido:e.nombre.apellido_p,
               resultados:set(tuple[nombre_cur:m.asignatura.curso.nombre,
                                    cod_asig:m.asignatura.num_asig,
                                    fecha:m.fecha,
                                    nota:m.nota])
              ]
from Estudiantes e, e.examinado_de m
where e.grupo = 'IG2'
```

7. Diseño físico de bases de datos orientadas a objetos

Al igual que en las bases de datos relacionales, el modelo lógico de datos de las BDOO es independiente de la localización física de los datos. Los datos también se almacenan en ficheros emplazados en diferentes dispositivos físicos que el administrador de la base de datos tiene que gestionar. Muchos de los conceptos de diseño físico de bases de datos relacionales se pueden aplicar para las BDOO. Sin embargo, hay ciertos cambios en la forma en que se realizan los agrupamientos y los índices de acceso a los datos, que deben ser estudiados. En esta sección se repasan los principales mecanismos que existen para diseñar el almacenamiento físico de los datos en BDOO.

7.1. Índices

Los principales tipos de estructuras de indexación que podemos encontrar en las BDOO son:

- **Índices de atributos**. Se usan para indexar los objetos de una clase según alguno de sus atributos. De esta manera se pueden recuperar rápidamente los objetos de una clase que cumplen cierta propiedad sobre el atributo indexado. Por ejemplo, se pueden indexar los objetos de la clase Empleados por su atributo dni.

- **Índices de objetos**. Se usan para indexar los objetos de una clase según su OID. Este índice contiene una entrada con el OID de cada objeto de la clase, y un puntero a su dirección de memoria. Esta alternativa resulta muy útil para navegar por los objetos ya que para pasar de un objeto de una clase a otro que referencia desde alguno de sus atributos se utiliza el índice. Para llegar al objeto referenciado solo hay que acceder al índice de la clase adecuada y encontrar el OID y la dirección de memoria de dicho objeto.

- **Índices de trayectoria**. En este caso el índice para una trayectoria dada consiste en una tabla con tantas columnas como componentes haya en la trayectoria o camino de acceso. La tabla contiene una fila por cada combinación de OID que determinen una ocurrencia de dicho camino dentro de la base de datos. Por ejemplo, supongamos la siguiente trayectoria p.imparte a, a.examinados e, e.estudiante.estudia d que indica los departamentos (d) en los que estudian los estudiantes (e) examinados de las asignaturas (a) de cada profesor (p). Supongamos también que esta trayectoria es muy importante y que tiene que ejecutarse lo más rápidamente posible. Definiendo un índice sobre este camino de acceso, podríamos tener la tabla 1.2 de relaciones entre los profesores, las asignaturas, los estudiantes y los departamentos almacenados en la base de datos.

p	a	e	d
oid.P1	*oid.A1*	*oid.E1*	*oid.D1*
oid.P1	*oid.A2*	*oid.E1*	*oid.D1*
oid.P1	*oid.A2*	*oid.E2*	*oid.D2*
oid.P1	*oid.A2*	*oid.E3*	*oid.D3*
oid.P1	*oid.A3*	*oid.E3*	*oid.D3*
oid.P2	*oid.A4*	*oid.E1*	*oid.D1*
oid.P2	*oid.A4*	*oid.E3*	*oid.D3*
oid.P2	*oid.A1*	*oid.E4*	*oid.D4*
oid.P3	*oid.A4*	*oid.E2*	*oid.D2*
oid.P4	*oid.A4*	*oid.E5*	*oid.D5*
oid.P4	*oid.A1*	*oid.E5*	*oid.D5*
oid.P4	*oid.A1*	*oid.E1*	*oid.D1*

Tabla 1.2. Ejemplo de índice de trayectoria

Para ejecutar la trayectoria desde una consulta, no será necesario acceder a la base de datos y recuperar el gran número de objetos por los que pasa. Por ejemplo, para conocer el departamento de los estudiantes a los que da clase el profesor con OID *oid.P4*, leyendo las filas de este índice se puede saber que solamente hay que recuperar los departamentos con OID *oid.D1* y *oid.D5*.

7.2. Agrupamientos

Las técnicas de agrupamiento se utilizan para posicionar los objetos en memoria secundaria en posiciones contiguas, de forma que el proceso de recuperación se acelere. Generalmente, en BDOO se pueden utilizar los siguientes tipos de agrupamientos:

- Por **clases**. Este suele ser el modo de posicionamiento por defecto. Todos los objetos de una clase se almacenan contiguos. Estos pueden incluir a los objetos de otras clases que sean especializaciones suyas (subclases). De esta manera, cuando se tiene una jerarquía de especialización simple expresada en forma de árbol cuyos nodos son las clases de la jerarquía, los objetos de las diferentes clases se almacenan según el orden definido por el recorrido en preorden de los nodos del árbol.

- Por **valores de los atributos.** Los objetos de una clase se agrupan por el valor de alguno de sus atributos. Por ejemplo, los objetos de la clase estudiante se pueden almacenar ordenados por apellidos. Si se combina este método con un índice sobre el mismo atributo, entonces la evaluación de las consultas que involucren a dicho atributo se puede ver muy favorecida.

- Por **relaciones de asociación.** Aquellos objetos que se relacionen con otros referenciándolos desde alguno de sus atributos, se almacenan de forma contigua en el mismo bloque de memoria, por lo que ambos objetos se recuperan al mismo tiempo. De esta manera, la recuperación de objetos que se relacionan con otros objetos se acelera mucho, ya que todos ellos se

recuperan en unas pocas operaciones de lectura. Por ejemplo, se puede recuperar rápidamente el objeto de un departamento junto con los objetos de todos sus profesores.

- Por **relaciones de agregación.** Aquellos objetos que se componen de otros objetos se pueden almacenar de manera que los componentes se agrupan incrustados en los compuestos (ver el ejemplo de la figura 1.10). De esta forma, con pocos accesos al disco es fácil recuperar un objeto junto con sus componentes.

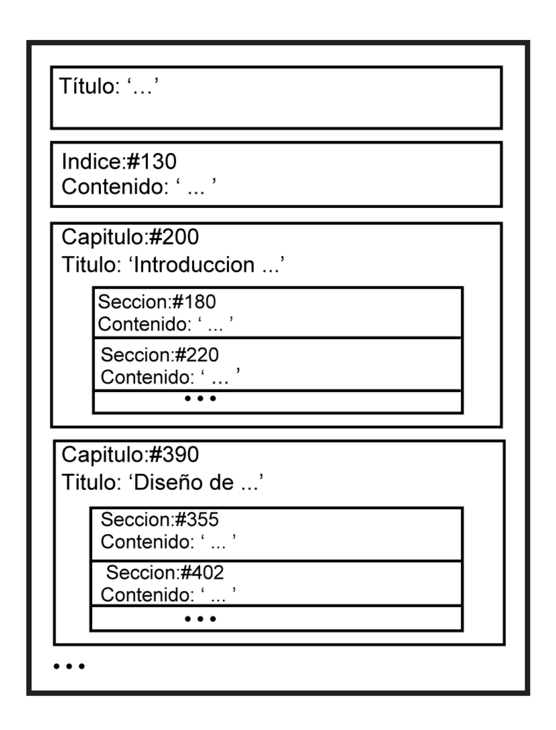


Figura 1.10. Ejemplo de agrupamiento por relaciones de agregación

Bibliografía

Atkinson, M. et al. (1989): *The Object-Oriented Database System Manifesto*, Proceedings of 1st International Conference on Deductive and Object-Oriented Databases.

Cattell, R. et al. (2000): *The Object Data Standard: ODMG 3.0*, Morgan Kaufmann.

Date, C. J. (2001): *Introducción a los Sistemas de Bases de Datos*, (7.ª edición), Prentice-Hall.

Elmasri, R., Navathe, S. (2011): *Fundamentals of Database Systems*, (6.ª edición), Addison-Wesley Longman.

Harrington, J. (2000): *Object Oriented Database Design Clearly Explained,* Morgan Kaufmann.

Silberschatz, A., Korth, H., Sudarshan, S. (2002): *Fundamentos de Bases de Datos*, (4.ª edición), McGraw Hill.

Ejercicios

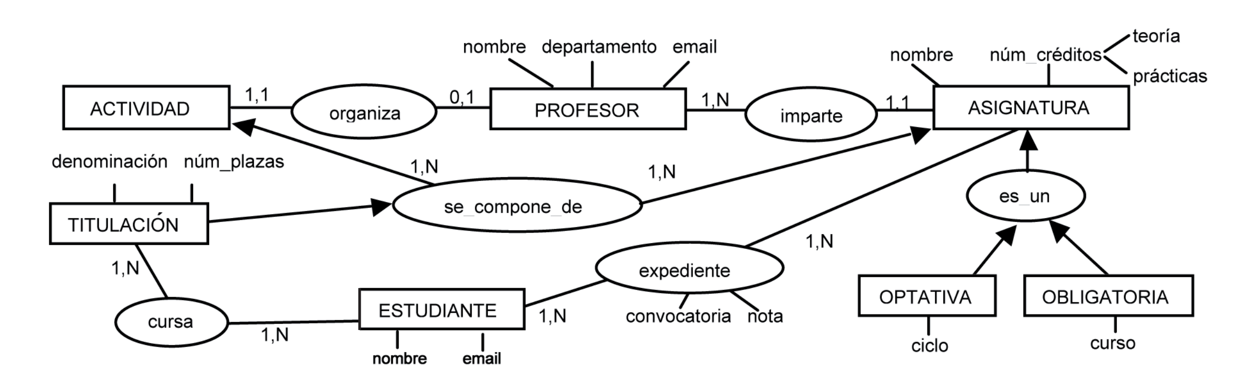
Ejercicio 1

Utilizando los siguientes conceptos, diseñar un diagrama conceptual que contenga al menos dos jerarquías de especialización y dos de agregación. El diagrama también deberá contener alguna relación de asociación. Dar atributos a todas las entidades.

- Periódicos
- Noticias
- Anuncios publicitarios
- Fotografías
- Textos
- Trabajadores de redacción
- Trabajadores de taller
- Redactores jefe
- Reporteros
- Maquetadores de taller
- Reveladores de taller
- Escribir noticia
- Aceptar noticia
- Maquetar noticia
- Maquetar anuncio
- Revelar fotografía

Ejercicio 2

Dado el siguiente esquema conceptual diseñar una base de datos orientada a objetos.

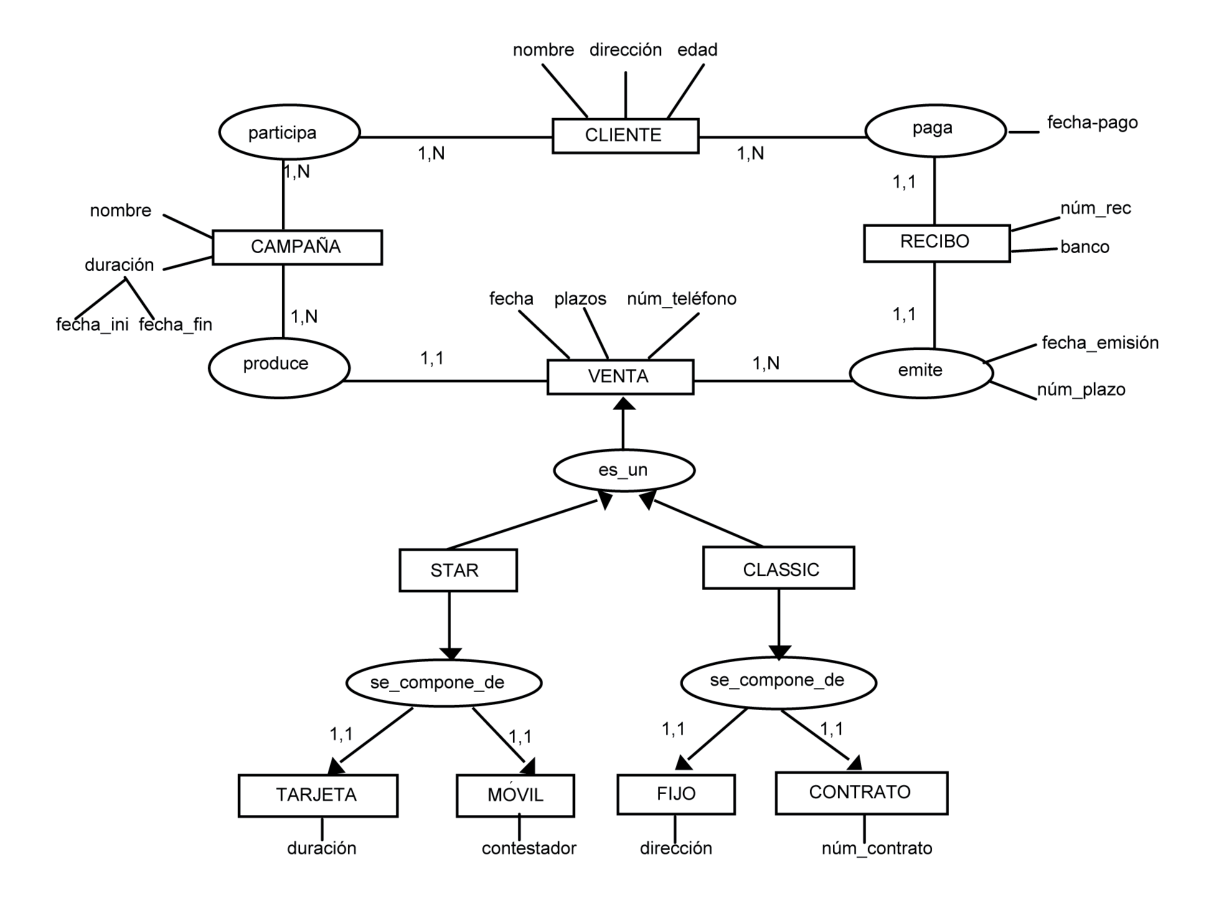


Formular las siguientes consultas sobre la base de datos OO anterior:

1. Utilizando la asociación expediente, seleccionar los profesores de las asignaturas de la titulación de 'Derecho' en las que se hayan matriculado más de 30 estudiantes.
2. Seleccionar los nombres de los estudiantes que siempre aprueban las asignaturas en primera convocatoria.

Ejercicio 3

Dado el siguiente esquema conceptual diseñar una base de datos orientada a objetos.

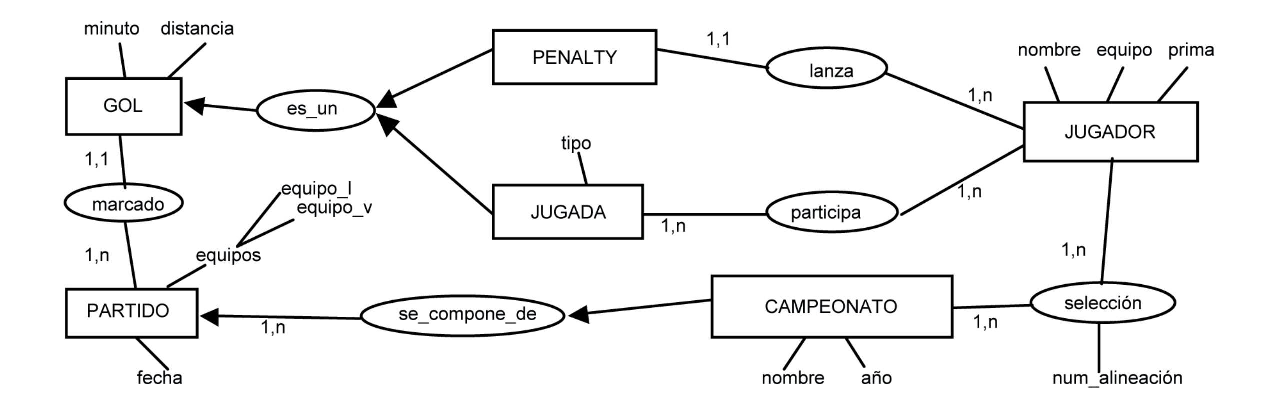


Formular las siguientes consultas sobre la base de datos OO anterior:

1. Para el cliente «Oscar Pérez Pérez», recuperar sus números de teléfono con su fecha de venta.
2. Recuperar la duración media de todas las tarjetas vendidas en el año 2010.
3. Para todos los clientes que hayan participado en la campaña llamada «talk-more», recuperar el nombre de aquellos que hayan pagado un recibo en la misma fecha en que fue emitido.
4. Recuperar todos los recibos del «Banco de Morella» ordenados por teléfonos.

Ejercicio 4

Dado el siguiente esquema conceptual diseñar una base de datos orientada a objetos.

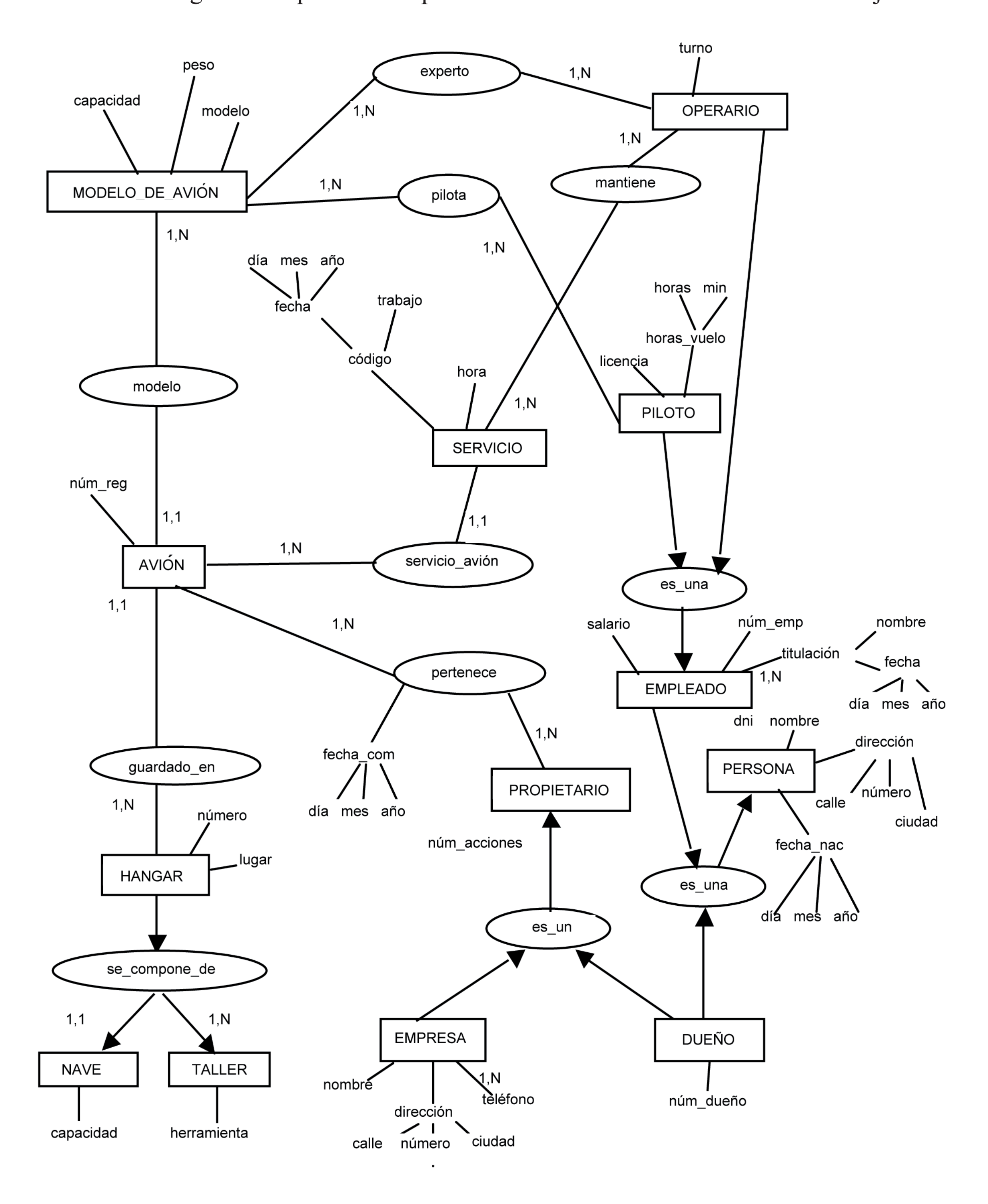


Formular las siguientes consultas sobre la base de datos OO anterior:

1. Seleccionar los jugadores que hayan lanzado más de 5 penaltis y que hayan participado en alguna jugada de gol tipo remate desde una distancia de más de 20 metros.
2. Seleccionar los nombres de los jugadores con una prima económica mayor.

Ejercicio 5

Dado el siguiente esquema conceptual diseñar una base de datos orientada a objetos.



Formular las siguientes consultas sobre la base de datos OO anterior:

1. Recuperar el nombre de los modelos de los aviones que posee la empresa llamada 'Aerosmith'.
2. Recuperar el nombre de cada piloto junto con un número que indique cuántos modelos de aviones es capaz de pilotar.
3. Recuperar los aviones que pertenecen a más de 3 propietarios.
4. Recuperar cuántos aviones que puede pilotar 'José Pérez' agrupados por modelos.
5. Recuperar las fechas y las horas de los servicios de los aviones que se guardan en el hangar número 7.
6. Visualizar el salario máximo y mínimo que un empleado puede tener.
7. Recuperar los nombres y las titulaciones de las parejas de operarios y pilotos que vivan en la misma calle de la misma ciudad.
8. Calcular el peso y la capacidad media de los aviones.
9. Recuperar el número y la posición de cada hangar cuya nave tenga una capacidad superior a los 1000 metros cuadrados y ordenar por la capacidad de la nave.
10. Recuperar los pilotos que siempre vuelan en aviones con capacidad superior a los 200 pasajeros.

CAPÍTULO 2

Sistemas de recuperación de información y documentos estructurados

1. Introducción

Los Sistemas de Recuperación de Información (SRI) tratan con la representación, almacenamiento y recuperación de documentos. Por definición, el objetivo general de un SRI es que dada una consulta proporcionada por el usuario el sistema recupere información relevante. En los SRI tradicionales cada documento se almacena en un fichero separado. Previamente, su contenido textual debe ser procesado para su indexación. Es precisamente este mecanismo de indexación el que se utiliza para procesar las consultas de los usuarios y recuperar los documentos relevantes.

Durante las últimas décadas han surgido nuevos formatos para representar documentos estructurados que permiten expresar los atributos y la organización de los documentos junto con sus contenidos. XML (*Extensible Markup Language*) es un formato universalmente aceptado como estándar de intercambio de datos y documentos que permite ser procesado, entre otras muchas cosas, para indexar su contenido y estructura. En este capítulo se estudian los principales modelos y técnicas que se utilizan en los sistemas de recuperación de información tradicionales y para documentos estructurados.

1.1. Objetivos de aprendizaje

Los objetivos de aprendizaje de este capítulo se enumeran a continuación:

a) Definir qué son los SRI y sus conceptos más importantes.
b) Diferenciar entre un sistema de bases de datos y un SRI en cuanto a su utilización y técnicas de desarrollo.
c) Explicar en qué consiste la tarea de recuperar información.
d) Definir los elementos de un SRI, sus funciones y relaciones.
e) Describir los principales modelos de representación de documentos.
f) Explicar los modelos de recuperación de información clásicos, sus características y propiedades.
g) Explicar el concepto de eficacia de un SRI y sus mecanismos básicos de medida.
h) Describir en qué consisten los principales mecanismos de especificación de consultas en SRI tradicionales.
i) Explicar los pasos del proceso de indexación y almacenamiento de documentos en un SRI.
j) Describir los principales tipos de índices de los SRI y su funcionamiento en la búsqueda y recuperación de información.
k) Explicar las peculiaridades de los SRI que almacenan documentos estructurados en cuanto a sus consultas y esquemas de indexación.

2. Sistemas de bases de datos versus sistemas de recuperación de información

A pesar de que los SRI no son tan conocidos como los sistemas de bases de datos (SBD), se están utilizando desde hace tantos años como los SBD más antiguos. Ambos tipos de sistemas deben permitir al usuario almacenar y recuperar información, sin embargo, hay diferencias sustanciales en su funcionamiento y ámbito de aplicación. En este apartado se estudian las principales diferencias entre estos dos tipos de sistemas de información.

Los SBD tradicionales solo pueden utilizarse para recuperar datos y sus técnicas no se pueden emplear en aplicaciones que necesiten rescatar documentos. Igualmente los SRI solo permiten almacenar documentos y no son adecuados para almacenar y recuperar datos. Por esta razón, desde sus orígenes, ambos tipos de sistemas han ido evolucionando separadamente, y sus técnicas y modelos son completamente diferentes. Sin embargo, durante los últimos años, los principales sistemas de gestión de bases de datos han ido incorporando módulos para el manejo de información textual que les permite indexar y gestionar documentos. De esta manera, ahora es posible combinar datos y documentos dentro de un mismo sistema de información, e incluso es posible realizar consultas que combinen condiciones de recuperación sobre ambos tipos de información.

Para almacenar datos dentro de una base de datos, previamente es necesario diseñar un esquema lógico que defina la estructura y el tipo de los datos que se van a insertar. Por ejemplo, en una base de datos relacional, esta información se especifica al crear las tablas donde se van a insertar los datos. Cada tabla se compone de un conjunto de columnas y cada columna almacena un solo dato del tipo correspondiente. Sin embargo, para los documentos de una colección no se puede crear un esquema de base de datos porque el contenido de cada documento es diferente. Sus secciones y párrafos tienen longitudes muy variadas y pueden llegar a ser muy largos. Además, es imposible predecir cuántas secciones o cuántos párrafos tendrá cada uno de los documentos de la colección. Por esta razón, en los SRI no se crea ninguna estructura de datos para almacenar los documentos. La unidad de almacenamiento y recuperación es el documento, y cada documento se almacena separadamente en un solo bloque, normalmente un solo fichero.

A la hora de especificar sus necesidades de información, los usuarios de un SRI tienen muchas más dificultades que con los SBD. Los datos de un SBD siempre tienen una estructura y un significado que vienen claramente definidos por el esquema de la base de datos que los almacena. Una consulta a un SBD en SQL define claramente unas condiciones de recuperación que todos los datos devueltos van a cumplir exactamente, con lo que siempre se puede decir que las respuestas satisfacen completamente las necesidades de los usuarios. Sin embargo, dadas las características de la información que manejan, los SRI no pueden ser tan exactos, y entre el conjunto de documentos que devuelven al usuario se pueden encontrar algunos que no son tan relevantes como otros. La principal razón de esta diferencia de comportamiento es que los SRI tratan con textos escritos en lenguaje natural, siendo este un

lenguaje cuya ambigüedad dificulta su manejo por los ordenadores, sobre todo por la imposibilidad que tienen de entender su significado preciso.

Por ejemplo, cuando se recupera de una base de datos el teléfono del Ayuntamiento de Peñíscola, el usuario realiza una consulta SQL como por ejemplo la siguiente:

```
select distinct telefono
from ayuntamientos
where municipio = 'Peñíscola'
```

Además, todos los elementos de la respuesta se corresponderán con la información solicitada. Sin embargo, para buscar esta información con un SRI, la consulta que se puede utilizar es mucho menos exacta. En este caso, un usuario podría indicar: (teléfono ayuntamiento Peñíscola).

Como respuesta del sistema, el usuario recibirá una lista de documentos ordenados según su capacidad de satisfacer las condiciones de la consulta. Cuanto mayor sea esta capacidad mayor será la relevancia del documento con respecto a la consulta. Sin embargo, a pesar de esto, muchos de los documentos de la respuesta no proporcionarán la información que se solicita, es decir, el número de teléfono del Ayuntamiento de Peñíscola.

Para realizar su función, los SRI procesan el contenido de los documentos que almacenan, y los representan por medio de un conjunto de palabras que se extraen de su contenido. Posteriormente, los documentos son considerados para su recuperación de acuerdo a su relevancia con respecto a la consulta inicial. Todo este proceso requiere la extracción de información de los textos y la utilización de esta información para evaluar la consulta. La dificultad no esta solo en cómo extraerla, sino en cómo utilizar la información para estimar la relevancia del documento con respecto a la consulta.

Como puede verse, la noción de relevancia es central en los SRI. De hecho, podemos decir que el principal objetivo de los SRI es recuperar el mayor número posible de documentos relevantes para el usuario, y recuperar el menor número posible de ellos que no sean suficientemente relevantes. En el resto del capítulo se estudian las principales características y técnicas empleadas por los SRI actuales. En la tabla 2.1 se resumen todos los conceptos explicados en este apartado.

Sistemas de Bases de Datos	Sistemas de Recuperación de Información
Almacenan datos con estructura regular y significado preciso	Almacenan documentos con estructura irregular y significado impreciso
Consultas con condiciones precisas	Consultas con condiciones aproximadas

`select distinct telefono`
`from ayuntamientos`
`where municipio = 'Peñíscola'`
	`(teléfono ayuntamiento Peñíscola)`

Resultados siempre relevantes	Resultados varían de muy relevantes a poco relevantes
Consultas completamente satisfechas	Consultas más o menos satisfechas

Tabla 2.1. Sistemas de bases de datos versus sistemas de recuperación de información

3. Visión general

En esta sección se presentan de una manera general los conceptos más básicos de los SRI que serán mejor explicados en el resto de secciones del capítulo. Estos conceptos sirven para comprender cómo funciona un SRI de una manera preliminar y permiten comenzar a profundizar en las cuestiones más específicas del capítulo como son los modelos de representación de los documentos, los mecanismos de evaluación de consultas y las técnicas de inserción, indexación y búsqueda de documentos.

3.1. La tarea de recuperar información

A la hora de utilizar un SRI, el usuario tiene que traducir sus necesidades de información en consultas escritas en el lenguaje o interfaz gráfica proporcionada por el sistema. En general, tal y como expresa la figura 2.1, puede decirse que la tarea de recuperar información es un proceso cíclico cuyos objetivos iniciales no están bien definidos y pueden ir cambiando a lo largo del proceso.

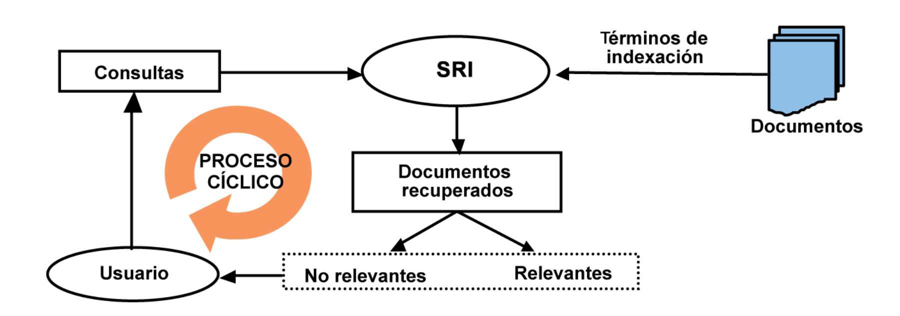


Figura 2.1. La tarea de recuperar información con un SRI

Normalmente, la tarea de recuperar información implica especificar un conjunto de palabras clave que describan los contenidos de los documentos a recuperar. Después de un primer intento y de observar los documentos recuperados, si la respuesta del sistema no se considera los suficientemente buena, el usuario suele entonces escoger otro conjunto de palabras clave para describir de nuevo la consulta anterior y tratar de obtener una respuesta que cubra mejor sus necesidades.

En otras ocasiones, al recibir la respuesta a una consulta, al usuario le puede surgir una nueva necesidad de información que cambia su objetivo inicial y que se expresa con una nueva consulta más o menos relacionada con la anterior. Por ejemplo, un usuario buscando información de los actores de una película puede descubrir el nombre de un actor que hasta entonces desconocía. Este hecho, le puede llevar a una nueva consulta en la que trate de encontrar más películas en las que haya participado dicho actor.

3.2. Arquitectura de un SRI

En la figura 2.2 se proporciona un esquema que expresa la arquitectura genérica de un SRI y su funcionamiento en el proceso de almacenamiento y recuperación de información. Nótese que los usuarios de un SRI van a realizar únicamente operaciones de consulta y las operaciones de almacenamiento de documentos las realizará su administrador.

El contenido de los documentos que se almacenan en un SRI se suele representar por medio de un conjunto de términos que sirven para su indexación. Conforme se van almacenando los documentos, estos se van procesando para extraer sus términos de indexación que son insertados en el índice del sistema. Un esquema de indexación, o simplemente índice, es una estructura de datos con información extraída tras procesar los documentos y que permiten su recuperación rápida tras comprobar que satisfacen las condiciones de las consultas.

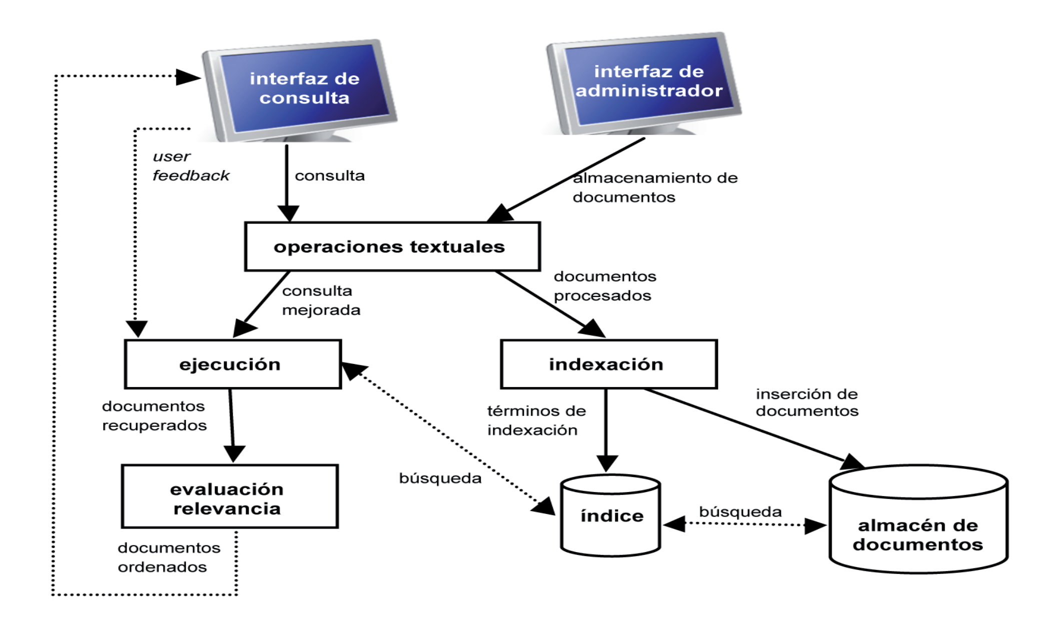


Figura 2.2. Arquitectura de un SRI

Al igual que el contenido de los documentos, las necesidades de información de los usuarios también se suelen representar con un conjunto de términos. El proceso de recuperación comienza analizando la consulta del usuario para trasformarla con operaciones textuales en otra consulta mejorada que devuelva información más relevante. A continuación, la consulta se ejecuta sobre el índice y el sistema recupera un conjunto de documentos. Antes de enviar estos documentos al usuario, deben ser ordenados según su relevancia con respecto a la consulta del usuario. A veces, después de visualizar los documentos, el usuario puede indicar cuáles son los que más le interesan (*user feedback*). Esta información puede ser utilizada por el sistema para reformular la consulta inicial y devolver al usuario una respuesta con más documentos relevantes. En el resto del capítulo se explican todos estos conceptos mucho más detalladamente.

4. Modelos de representación de documentos

Para representar los documentos almacenados, cada SRI utiliza su propio modelo de documento. En el modelo de representación de documentos se define qué información se va representar acerca de cada documento almacenado. Por eso, el modelo de documento utilizado por un SRI va a tener consecuencias en cuestiones importantes como el formato y las estructuras de datos que se van a utilizar para almacenar los documentos, y los tipos de condiciones que se van a poder especificar en los lenguajes de recuperación del sistema.

Algunos modelos de documento solamente representan su contenido textual, en otros casos junto con el texto de los documentos se representan otras propiedades de los mismos, y en algunos sistemas se pueden utilizar lenguajes de marcado para representar el contenido de los documentos junto con su organización y atributos. En esta sección se explican las diferentes alternativas que existen para crear un modelo de documento.

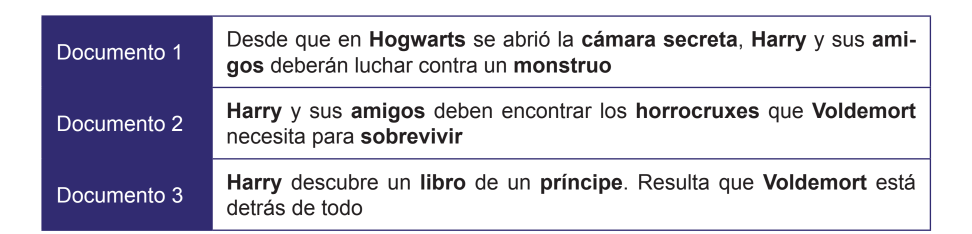
4.1. Representación del contenido textual

En todos los SRI se representa el contenido textual de los documentos. Se considera que cada documento está descrito por un conjunto de *palabras clave* que resumen su contenido, las cuales son también denominadas *términos de indexación*, ya que se insertan en un índice que se utiliza para procesar las consultas. Para seleccionar los términos de indexación de un documento, este tiene que ser procesado por el sistema con el fin de identificar las palabras que son realmente importantes para describir su contenido. Sin embargo, algunos sistemas realizan una indexación *full-text* de los documentos que almacenan, lo cual significa indexar todas las palabras del texto. Esto resulta en un modelo de documento muy completo pero con demasiado costo computacional. Por eso, para grandes colecciones de documentos, es mejor utilizar los sistemas que son capaces de extraer automáticamente las palabras del texto que tengan un mayor significado. Durante este proceso primero se eliminan las *stopwords* (palabras sin significado como artículos, pronombres, preposiciones, etc.), después se extraen las raíces gramaticales que forman conjuntos de palabras con el mismo significado (*stemming*), y finalmente se identifican las frases nominales (se agrupan las palabras en conceptos y se eliminan los adjetivos, adverbios y los verbos).

Todas estas operaciones sobre el texto reducen la complejidad del modelo de representación de los documentos, y proporcionan un buen conjunto de términos de indexación para el documento. Sin embargo, es importante tener en cuenta que ninguna de ellas resuelve los problemas de ambigüedad del lenguaje causados entre otras cosas por la utilización de palabras con varios significados (polisemias) o varias palabras con el mismo significado (sinónimas). Aunque se han estudiado mucho, los problemas de ambigüedad del lenguaje siguen sin estar resueltos, así que se puede decir que es imposible expresar el contenido de los documentos de manera exacta.

4.1.1. Matriz de términos/documentos

Para representar el contenido textual de los documentos, en muchos SRI se utiliza una estructura de datos consistente en una matriz con dos dimensiones denominada *matriz de términos/documentos*. En esta matriz se almacenan los términos de indexación que se han extraído tras procesar los documentos para almacenarlos en el sistema. Más específicamente, en una dimensión de esta matriz se consideran todos los documentos almacenados en el sistema y en la otra todos los términos de indexación que describen su contenido. En cada posición de la matriz se almacena un valor que indica si el documento contiene dicho término, pudiendo ser un valor booleano de cierto/falso o un número real que exprese la importancia que tiene dicho término en el documento. Toda esta información será utilizada por el SRI para procesar las consultas.



Documento 1	Desde que en **Hogwarts** se abrió la **cámara secreta**, **Harry** y sus **amigos** deberán luchar contra un **monstruo**
Documento 2	**Harry** y sus **amigos** deben encontrar los **horrocruxes** que **Voldemort** necesita para **sobrevivir**
Documento 3	**Harry** descubre un **libro** de un **príncipe**. Resulta que **Voldemort** está detrás de todo

Tabla 2.2. Ejemplos de documentos con términos de indexación

Por ejemplo, considerando los tres documentos de la tabla 2.2 en el que los términos de indexación aparecen marcados en negrita, se puede construir la matriz de la tabla 2.3. En esta tabla, cada posición de la matriz almacena un valor booleano indicando si el texto contiene al término.

Término	Doc. 1	Doc. 2	Doc. 3
hogwarts	1	0	0
harry	1	1	1
amigos	1	1	0
cámara secreta	1	0	0
monstruo	1	0	0
voldemort	0	1	1
horrocruxes	0	1	0
sobrevivir	0	1	0
príncipe	0	0	1
libro	0	0	1

Tabla 2.3. Ejemplo de matriz de términos/documentos

4.2. Metadatos

En algunos SRI se asigna a cada documento almacenado algunos metadatos para luego permitir a los usuarios incluir en sus consultas condiciones sobre ellos. Los metadatos se definen literalmente como datos acerca de los datos, en el caso de un SRI son datos referentes a las propiedades y atributos de los documentos. En la mayoría de los SRI tradicionales, los metadatos se almacenan en registros de bases de datos y ficheros separados de los documentos. Sin embargo, tal y como veremos en el siguiente apartado, existen formatos como XML que permiten representar la estructura y atributos de los documentos junto a su contenido.

Algunos ejemplos frecuentes de metadatos asociados a los documentos son su título, autor, fecha de publicación, editorial, número de páginas, etc. Las palabras que describen el contenido de una fotografía también se consideran metadatos. Por ejemplo, en la figura 2.3 se presenta la interfaz de búsqueda avanzada de imágenes en Google. Como puede verse se pueden especificar condiciones de recuperación sobre muchos tipos diferentes de metadatos. Los metadatos no pertenecen propiamente al contenido del documento como lo hacen sus figuras o párrafos, sino que lo que hacen es considerar las propiedades y los atributos del documento para darnos información adicional sobre el mismo.

Actualmente se han definido muchos formatos estándar de definición de metadatos para documentos. Algunos se especializan en documentos sobre un tema, una profesión, o de cierto tipo. Por ejemplo, en su cabecera los documentos HTML aceptan la etiqueta <meta> que sirve para insertar metadatos. El más famoso estándar de metadatos se denomina Dublin Core y consiste en un conjunto de elementos para describir una amplia gama de recursos de red. También existen vocabularios expresamente creados para describir el contenido de los documentos médicos, y lenguajes estándar para expresar las características de las noticias publicadas en los periódicos.

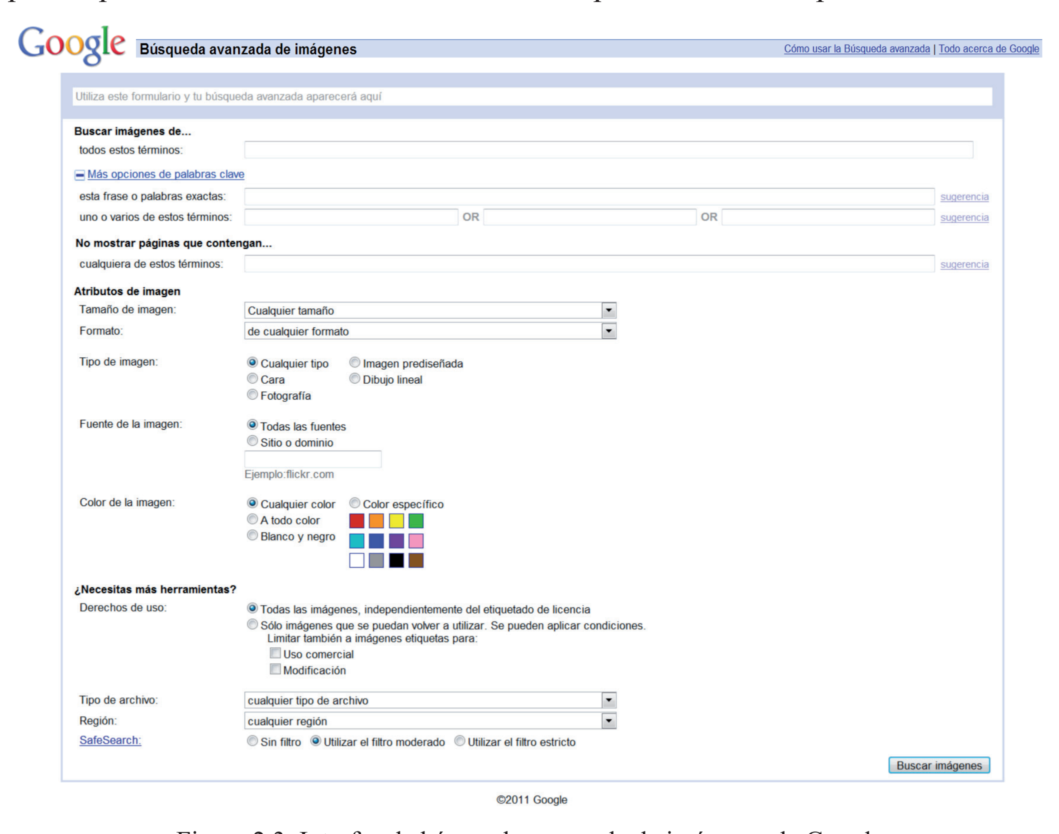


Figura 2.3. Interfaz de búsqueda avanzada de imágenes de Google

4.3. Documentos estructurados

Hoy en día existen muchos documentos que tienen su estructura implícitamente definida junto a su contenido por medio de algún lenguaje estándar de marcado como pueden ser HTML o XML. Estos documentos pueden encontrarse tanto en SRI como en la web. De hecho, la manera en que funciona un buscador web como Google o Yahoo! no se diferencia mucho de lo que podría ser la interfaz de usuario de un SRI típico.

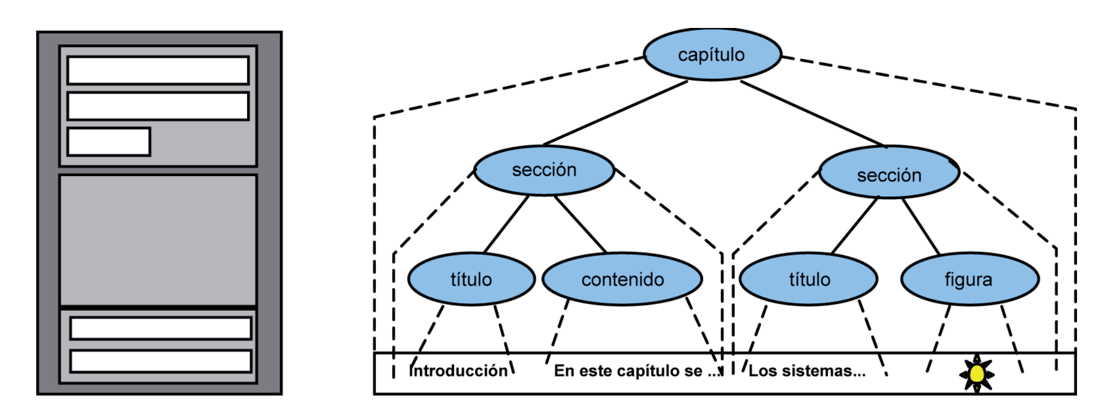


Figura 2.4. Estructura de un documento

Algunos SRI que almacenan documentos estructurados definen una estructura fija que deben seguir todos los documentos (ver la parte izquierda de la figura 2.4). Estos sistemas sirven para almacenar algunos tipos de documentos que siempre se organizan de la misma manera, como por ejemplo los formularios. Sin embargo, la mayoría de documentos no tienen una estructura plana y rígida, sino todo lo contrario. De unos documentos a otros las estructuras presentan muchas variaciones, que además son complejas y con muchas relaciones de anidamiento. Por ejemplo, un documento de tipo libro tiene un número indeterminado de capítulos con varias secciones que se componen a su vez de otras subsecciones con las mismas características. Generalmente, como se muestra en la parte derecha de la figura 2.4, la estructura de un documento es jerárquica formando un árbol de nodos cuyas hojas almacenan el contenido del documento.

Los documentos estructurados tienen su organización lógica implícitamente representada junto con su contenido. Como muestra el documento ejemplo de la figura 2.5, para delimitar las componentes del documento se utilizan unas etiquetas que marcan su comienzo y su final. Además, es posible utilizar atributos con metadatos acerca de las componentes. Entre otras muchas, estos lenguajes de marcado tienen las ventajas de que son fácilmente manipulables y permiten procesar el contenido de documento junto con su estructura. Por esta razón, antes de ser almacenados en un SRI, los documentos estructurados pueden ser procesados para extraer sus términos de indexación junto con otros datos adicionales sobre su estructura.

```
<SigmodRecord>
  <issues>
    <volume>1</volume>
    <number>1</number>
      <articles>
          <articlesTuple>
            <title>Contents, Sketches, Publications, Interesting Happenings.</title>
            <initPage> </initPage>
            <endPage> </endPage>
            <authors> </authors>
          </articlesTuple>
          <articlesTuple>
            <title>Chairman's Message.</title>
            <initPage>1</initPage>
            <endPage>1</endPage>
            <authors>
                <author AuthorPosition="002">Donald J. Hatfield</author>
                <author AuthorPosition="001">Kendall R. Wright</author>
            </authors>
          </articlesTuple>
      </articles>
  </issues>
</SigmodRecord>
```

Figura 2.5. Ejemplo de documento XML

Cada vez hay más SRI que proporcionan mecanismos para especificar consultas que combinan condiciones sobre el contenido y atributos de los documentos con condiciones sobre la estructura de los mismos. De esta manera es posible recuperar documentos que contengan cierta palabra clave en su título o en su resumen, o encontrar documentos que, por ejemplo, contengan cierta palabra clave en más de tres secciones, o documentos con una sección de introducción y otra de referencias, etc.

Desde el punto de vista de los usuarios, resulta muy interesante poder utilizar la información sobre la estructura de los documentos para mejorar el funcionamiento de los SRI. Está demostrado que las consultas que contienen condiciones de recuperación sobre la estructura y el contenido de los documentos permiten describir mejor los requerimientos de usuario. Además, la posibilidad de recuperar solamente aquella porción de los documentos que realmente interesa como puede ser el resumen o la sección de referencias, abre un nuevo abanico de posibles aplicaciones para los SRI, que ya no se tienen que limitar a recuperar documentos completos.

4.3.1. Introducción al lenguaje XML

En esta sección se describen los conceptos más básicos del lenguaje XML (*Extensible Markup Language*) que es el formato más conocido para representar documentos estructurados. XML es un lenguaje de marcado de documentos, lo cual significa que añade al documento etiquetas que especifican su estructura y componentes. Entre otras muchas cosas, estas etiquetas sirven para poder interpretar el contenido de los documentos. Por ejemplo, considera el siguiente documento:

```
<temperatura>37,5</temperatura>
<hora>16</hora><minutos>39</minutos>
```

Las etiquetas entre los símbolos < y > describen el texto que contienen y facilitan la interpretación de su significado. El lenguaje XML permite definir con precisión estas etiquetas. XML ha sido estandarizado por el WWW Consortium (http://www.w3.org/XML/), y fue creado a partir de un estándar anterior mucho más complejo (SGML, *Standard Generalized Markup Language*), cuya aplicación más conocida es el lenguaje HTML con el que se escriben las páginas web.

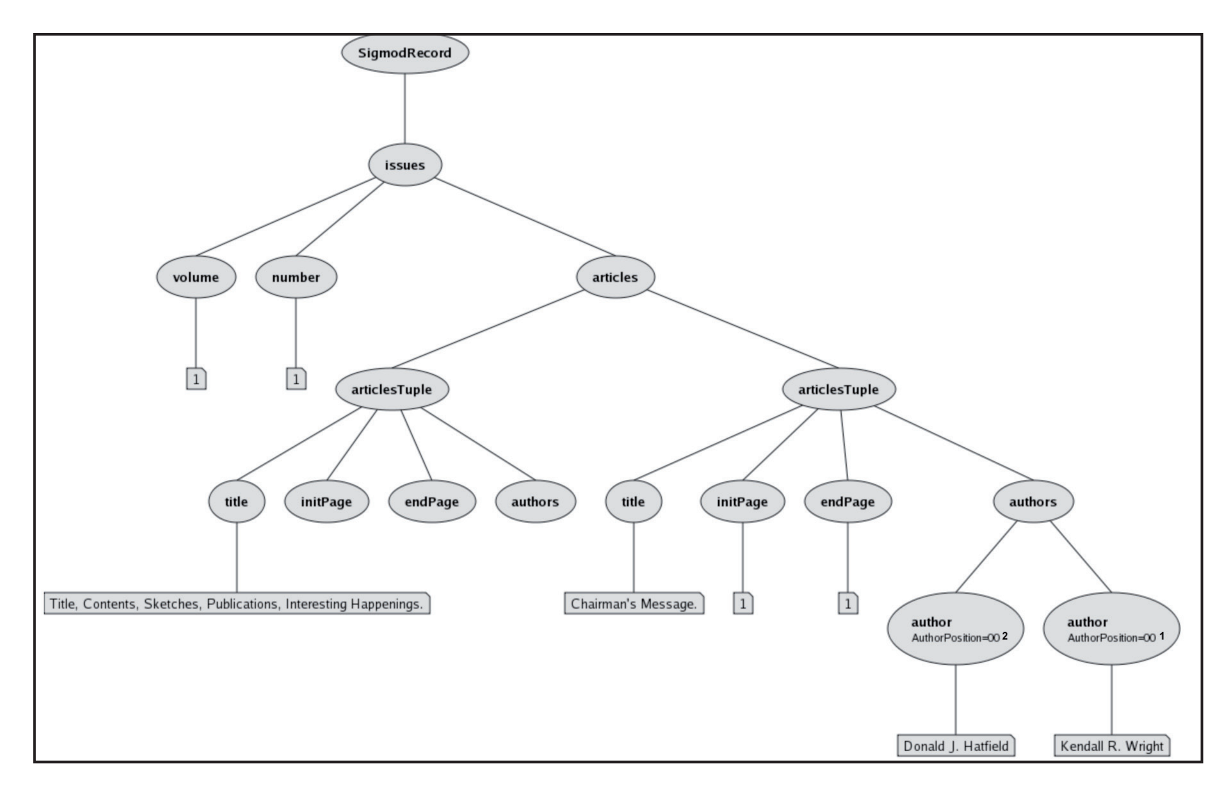


Figura 2.6. Modelo de documento de XML

En el modelo de documento de XML, las etiquetas pueden anidarse unas dentro de otras. Por ejemplo, la figura 2.5 muestra un documento XML con información sobre los ejemplares de la revista de artículos científicos *Sigmod Record*. Cada ejemplar viene marcado por las etiquetas <issues> ... </issues> y tiene un volumen, un número y los datos de varios artículos encerrados entre las etiquetas <articles> ... </articles>.

El documento de la figura 2.5 se puede representar gráficamente como muestra la figura 2.6. Como puede observarse, las relaciones de anidamiento de las etiquetas XML definen una estructura con forma de árbol que denominaremos *árbol de composición del documento*. Con el modelo de documento de XML, el contenido de los documentos siempre se almacena en las hojas de su árbol de composición. Los nodos internos del árbol se corresponden con las componentes complejas del documento. Una componente compleja no tiene contenido propio sino que sus contenidos se corresponden con la composición de los contenidos de las hojas del subárbol cuya raíz es dicha componente compleja. Además, todos los elementos de un documento pueden tener atributos cuyos contenidos no forman parte de los

contenidos del documento, sino que son metadatos que los describen. En nuestro ejemplo, cada elemento Author tiene un atributo denominado AuthorPosition para indicar el orden de los autores de un artículo.

5. Modelos de recuperación de información

El principal objetivo de un SRI es determinar qué documentos pueden ser relevantes para el usuario y cuáles no. Esta decisión la suele tomar un algoritmo que ordena los documentos recuperados según su relevancia y que opera de acuerdo a unas premisas básicas sobre el concepto de relevancia de documento. Dependiendo de cuáles son estas premisas diremos que el SRI sigue un modelo de recuperación u otro.

En general, un modelo de recuperación de información define cómo representar la información de los documentos y las consultas de los usuarios, e incluye una función que permite ordenar los documentos recuperados según su relevancia. Aunque existen muchos, destacan tres modelos clásicos de recuperación de información (booleano, vectorial y probabilístico) que han servido para sentar las bases de los actuales SRI. En esta sección se describen brevemente los conceptos fundamentales de estos tres modelos clásicos y también del modelo de recuperación que más se utiliza en la web, denominado *PageRank*™.

5.1. Modelo booleano

Con el modelo booleano los documentos se describen por medio de un conjunto de términos de indexación. Una consulta puede ser cualquier expresión del álgebra booleana sobre los términos de indexación. El conjunto de documentos recuperados por el sistema serán aquellos documentos cuyos términos de indexación cumplan la consulta.

Veámoslo por medio de un ejemplo. Sea la consulta [$q = k_a$ AND (k_b OR (BUT k_c))] que intenta recuperar los documentos que contienen el término k_a y el termino k_b, o que contienen el término k_a y no contienen a k_c. Según el modelo booleano, todos los documentos que cumplan esta condición serán devueltos al usuario con el mismo grado de relevancia, el cual no depende del número de veces que cada documento incluya los términos de la consulta ni de su orden de aparición. Si hay algún documento que cumpla esta condición de forma parcial, por ejemplo que contenga el término k_a pero no cumpla ninguna de las otras dos propiedades no aparecerá en la respuesta. Además, independientemente de si cumplen la condición k_b o la condición (BUT k_c) o ambas, todos los documentos de la respuesta tendrán la misma relevancia.

Este modelo ha sido muy utilizado porque es muy sencillo de implementar sobre un índice de términos o en una matriz de términos/documentos. Su principal desventaja es que no calcula un grado de relevancia para los documentos de la respuesta, ya que simplemente devuelve los documentos que cumplen las con-

diciones de la consulta. Se han hecho muchas extensiones y modificaciones al modelo booleano para crear nuevos modelos de recuperación de información que no tengan este inconveniente.

Otro inconveniente es que los usuarios inexpertos pueden formular consultas que devuelvan muy pocos documentos, por ejemplo poniendo demasiadas operaciones AND. Otras veces, las consultas pueden incluir demasiadas operaciones OR o pueden hacer una mala utilización de la negación, por lo que recuperarán demasiados documentos. Por ejemplo, la consulta [q = BUT k_a] devolvería todos los documentos almacenados menos los que contienen el término k_a.

5.2. Modelo vectorial

En este modelo clásico de recuperación de información también se considera cada documento descrito por un conjunto de términos de indexación que resumen su contenido. Además, en el modelo vectorial se asigna un peso a cada término de indexación para indicar su importancia al describir los contenidos de ese documento. Igualmente, cada consulta de usuario consiste en un conjunto de términos y un peso para cada uno de ellos indicando la importancia de ese término en la consulta. Todos estos pesos son utilizados por el sistema para calcular la relevancia de cada documento en la respuesta. A continuación, vemos una definición más formal del modelo vectorial.

Sea t el número de términos de indexación del índice de un SRI, sea $K = \{ k_1, k_2, ..., k_t \}$ su conjunto de términos de indexación y sea k_i un término de indexación cualquiera. Para indexar un documento d_j, se asocia un peso $w_{i,j} > 0$ con cada término de indexación k_i que aparece en el documento. Para un término que no aparece en el documento se tiene $w_{i,j} = 0$. Es decir, con el modelo vectorial cada documento d_j se representa por medio de un vector de términos de indexación $d_j = (w_{1,j}, w_{2,j}, ..., w_{t,j})$, y cada consulta de usuario q se representa con otro vector de la forma $q = (w_{1,q}, w_{2,q}, ..., w_{t,q})$. El modelo vectorial no especifica cómo establecer estos pesos, por lo que cada implementación sigue su propio método de estimación. El modelo *tf.idf* que veremos en la siguiente sección, es uno de los métodos más comúnmente utilizados para asignar pesos a los términos de indexación de los documentos.

Para evaluar la relevancia de un documento con respecto a una consulta, el modelo vectorial propone medir el grado de similitud de los vectores que los representan, o lo que es lo mismo, calcular la distancia que los separa en un espacio vectorial de t dimensiones, siendo t el número de términos de indexación del SRI. Para calcular esta distancia se puede utilizar como medida el coseno del ángulo que forman ambos vectores, el cual se obtiene dividiendo su producto escalar por el producto de sus normas.

d_j θ $\cos(\theta)$ q

$$\text{sim}(d_j, q) = \cos(\theta) = \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \times |\vec{q}|} = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{j=1}^{t} w_{i,q}^2}}$$

Dado que los pesos $w_{i,j}$ son siempre positivos y que el cos(0º) = 1 y cos(90º) = 0 el grado de similitud siempre es un valor entre 0 y 1. De esta manera, con esta medida es posible devolver al usuario un conjunto de documentos ordenados por relevancia. Lo normal es establecer una relevancia mínima, por ejemplo 0,6, y así todos los documentos cuya similitud con la consulta supere dicho mínimo serán recuperados.

El modelo vectorial es el más utilizado de todos ya que proporciona unas respuestas que se ajustan mucho a las necesidades de los usuarios, y una medida que permite ordenar los documentos por relevancia. Además, se puede implementar fácilmente con una matriz de términos/documentos en la que se almacenen los pesos de los términos de indexación de cada documento.

5.2.1. Estimación de los pesos: el modelo tf.idf

Cada término de indexación puede ser más o menos representativo de los contenidos de un documento. En general, un término que aparece muchas veces dentro de un documento será más representativo de sus contenidos que un término que solo aparece una vez. Sin embargo, un término que aparece en muchos documentos diferentes en realidad no sirve para distinguir unos de otros (por ejemplo, la palabra sistema), mientras que un término que aparece en menos documentos es más representativo de sus contenidos, ya que los identifica mejor (por ejemplo, el nombre de una persona famosa). En el modelo vectorial esta propiedad se representa por medio de los pesos que se asignan a los términos de indexación.

El modelo *tf.idf* es uno de los métodos más comúnmente utilizados para asignar un peso a cada término de indexación de un documento. En él se combinan dos medidas:

- la frecuencia del término (*tf*) que mide el grado de repetición del término dentro del documento,
- y la frecuencia inversa de documento (*idf*) que mide de manera inversa el número de documentos almacenados que contienen dicho término.

Como puede verse, esta medida da un mayor peso a los términos que se repiten en los documentos y que aparecen en pocos documentos. La siguiente expresión calcula un valor para el peso $w_{i,j}$ asociado con el termino k_i en el documento d_j. En ella, el valor de $tf_{i,j}$ viene determinado por la frecuencia de aparición del término k_i en el documento d_j y el valor de $idf_{i,}$ por la división entre el total de documentos almacenados (N) y el número de documentos donde aparece el término k_i (n_i). El logaritmo se utiliza para suavizar el efecto de este valor.

$$w_{i,j} = tf_{i,j} \cdot idf_{i,} = tf_{i,j} \cdot \log \left[\frac{N}{n_i} \right]$$

5.3. Modelo probabilístico

Al igual que en el modelo vectorial, en el modelo probabilístico tanto los documentos dentro del SRI como las consultas de los usuarios se modelan por medio de un conjunto de términos de indexación. Este modelo considera que para cada consulta hay un conjunto de documentos que contienen exactamente la información que el usuario necesita. Este conjunto de documentos es la respuesta ideal, y tanto el sistema como el usuario desconocen qué documentos lo forman.

Con el modelo probabilístico, el problema de recuperar información es un proceso iterativo que el sistema ejecuta automáticamente para tratar de aproximar su respuesta a la ideal. Su objetivo es encontrar las propiedades de los documentos de la respuesta ideal, es decir, encontrar los términos de indexación que hay que utilizar en la consulta que recupera los documentos de la respuesta ideal.

Inicialmente el usuario formulará una consulta cuya respuesta se aproximará más o menos a la respuesta ideal. A continuación, los términos de indexación de los documentos más relevantes serán utilizados por el sistema para reformular la consulta automáticamente y producir una nueva respuesta. Repitiendo este proceso tantas veces como sea necesario, el sistema irá construyendo una consulta cuya respuesta, aunque no sea la ideal, sea lo suficientemente buena para el usuario.

En el modelo probabilístico la relevancia de un documento con respecto a una consulta se define como la probabilidad de que el documento sea relevante para la consulta. El modelo define que la similitud entre un documento d_j y una consulta q se calcula por medio de la siguiente medida de similitud:

$$\mathrm{sim}(d_j, q) = \frac{probabilidad(d_j \text{ sea relevante para } q)}{probabilidad(d_j \text{ no sea relevante para } q)}$$

Un documento solamente se considera relevante para una consulta si su similitud es superior a 1, es decir, si la probabilidad de que el documento sea relevante para la consulta es mayor a la de no serlo. Una vez se han recuperado los documentos relevantes, esta medida de similitud también se utiliza para ordenarlos por relevancia antes de dárselos al usuario.

Sin embargo, el modelo probabilístico no define cómo calcular las probabilidades de la medida de similitud, por lo que se suelen estimar analizando la distribución de los términos de indexación por los documentos de la respuesta. Es decir, se considera que la probabilidad de que un término pertenezca a un documento relevante y la probabilidad de que pertenezca a un documento no relevante pueden aproximarse usando la distribución del término en los documentos relevantes y no relevantes recuperados por el sistema. Sea R el conjunto de documentos relevantes recuperados por una consulta, R_i el subconjunto de dichos documentos que contienen el término t_i, n_i el número de documentos almacenados en el sri que contienen el término t_i, y N el número total de documentos almacenados, entonces se define:

$$probabilidad(t_i \text{ pertenezca a un documento relevante}) = \frac{|R_i|}{|R|}$$

$$probabilidad(t_i \text{ pertenezca a un documento no relevante}) = \frac{n_i - |R_i|}{N - |R|}$$

De esta manera, la probabilidad de que un documento sea relevante para una consulta se puede estimar sumando las probabilidades de que los términos que contiene también lo sean, y viceversa, la probabilidad de que un documento no sea relevante para una consulta se puede estimar sumando la probabilidad de que los términos que contiene tampoco lo sean.

En la primera ejecución de una consulta, cuando todavía no se tiene ninguna información sobre la relevancia de los documentos, la estimación de estas probabilidades es especialmente complicada. Como solución, se puede suponer que los términos de indexación se distribuyen por los documentos según algún modelo de distribución (uniforme, normal, binomial, etc.). En las siguientes iteraciones, para decidir qué documentos son relevantes y cuáles no, se pueden utilizar dos métodos diferentes. Con el primer método se asume la hipótesis de que los primeros documentos devueltos son los más relevantes. El segundo método consiste en que el usuario vaya indicado cuáles de los documentos devueltos son más relevantes y cuáles menos (*user feedback*).

Al igual que el modelo vectorial, la principal ventaja del modelo probabilístico es que devuelve los documentos ordenados por relevancia de acuerdo a la medida de similitud. Entre los inconvenientes del modelo probabilístico, encontramos que no asigna pesos a los términos de indexación en base a su frecuencia u ordenación, lo cual se sabe que siempre conduce a respuestas menos precisas. Para resolver este inconveniente se ha formulado el modelo probabilístico BM25, el cual incorpora parámetros sobre la frecuencia de los términos en los documentos y en las consultas.

5.4. *PageRank*™

Hoy en día la web constituye el mayor SRI de todos los tiempos, ya que se trata de un enorme almacén de documentos. Los documentos que se encuentran en la web se diferencian en su idioma, vocabulario, formato (texto, HTML, PDF, imágenes, sonidos), etc. La principal diferencia entre la web y los SRI tradicionales es que la web es una enorme colección de documentos extremadamente heterogéneos que se van añadiendo dinámicamente y sin ningún control. Otra diferencia importante es que en la web los enlaces entre páginas son esenciales para encontrar información, mientras que ese es un concepto que no existe en un SRI tradicional.

Desde el punto de vista de los usuarios, la web es un enorme SRI donde para recuperar información disponen de buscadores como Google y Yahoo! Cada buscador sigue su propio modelo de recuperación de información, siendo la escalabilidad un

tema crucial, dada la gran magnitud del número de documentos a considerar por cada consulta de usuario. En esta sección se resumen los principales conceptos de *PageRank*™, el modelo de recuperación de información utilizado por Google para calcular la relevancia de una página web con respecto a una consulta de usuario.

Al igual que en los modelos clásicos, en el modelo de *PageRank*™ también se considera cada documento descrito por un conjunto de términos de indexación que resumen su contenido, y las páginas web recuperadas cumplen las condiciones de recuperación de la consulta. Pero además, en el modelo de *PageRank*™, cada página web lleva asociada una medida de calidad que sirve para determinar su relevancia. Esta medida es utilizada por Google para elaborar el ranking de páginas web que se muestra como respuesta a una consulta de usuario.

Intuitivamente, en el modelo de *PageRank*™ se considera que una página web es muy relevante si la suma de las relevancias de las páginas que la apuntan es también muy alta. Esta condición aporta mayor relevancia a las páginas web que son accesibles desde muchas otras páginas web, pero también a las páginas web que pueden ser accedidas desde menos páginas siendo algunas de ellas muy relevantes. Además, esta propiedad se propaga de unas páginas a otras, de tal manera que puede decirse que las páginas web más relevantes son aquellas que pueden ser accedidas siguiendo una o varias cadenas de enlaces en las que participan páginas web que también son relevantes. Por ejemplo, consideremos que la página web más relevante para la consulta «Universidad de Castellón» es la web de la Universitat Jaume I, www.uji.es. Para obtener ese resultado, el algoritmo *PageRank*™ se basa en que hay muchas páginas web a través de las cuales es posible llegar a la página web de la Universitat Jaume I y además, algunas de estas páginas son también muy relevantes.

Más formalmente, sea A una página web, N el número total de páginas en la web y $M(T_i)$ el número de páginas web T_i que contienen referencias a la página A. Si definimos $PR(T_i)$ como la relevancia (*PageRank*™) de la página T_i, entonces se calcula la relevancia de A con la siguiente formula:

$$PR(A) = \left(\frac{1-d}{N}\right) + d\left(\sum_{i=1}^{M(T_i)} \frac{PR(T_i)}{C(T_i)}\right)$$

En esta expresión $C(T_i)$ es el número de enlaces que salen de la página T_i y se utiliza para repartir la relevancia de una página entre las páginas a las que apunta con sus enlaces. Además, d es un factor variable que puede estar entre 0 y 1 y que se utiliza para representar la probabilidad de que un navegante utilice los enlaces de una página para seguir navegando por la web. Algunos expertos aseguran que el mejor valor de la variable d suele ser 0,85, aunque no se conoce cuál es el valor realmente utilizado por Google. La introducción de este factor en la fórmula resta algo de importancia a todas las páginas de Internet pero impide que las páginas que no tienen enlaces a ninguna otra página salgan especialmente beneficiadas.

Como puede observarse, esta formula es recursiva y necesita varias iteraciones para converger a un valor estable, pudiendo tener los *PageRank*™ iniciales cualquier valor asignado. Periódicamente, Google reconstruye sus índices y recalcula la relevancia de las páginas web para ajustarse mejor a la realidad. Lógicamente, al incrementarse el número de páginas en la web los *PageRank*™ son cada vez menores.

6. Evaluación de un SRI

Cuando se evalúa la eficiencia de un sistema de información se tienen en consideración parámetros como el tiempo y el espacio que requiere el sistema para ejecutar cada tipo de consulta. Sin embargo, en un SRI además de medir la eficiencia hacen falta otros parámetros que midan la eficacia del sistema con respecto a las consultas que se le formulan. La eficacia de un SRI puede definirse como su capacidad para recuperar los documentos que son relevantes para el usuario sin recuperar aquellos que no lo son. Esta medida de eficacia es muy importante porque los SRI no devuelven respuestas exactas.

Como expresa la figura 2.7, normalmente el conjunto de documentos relevantes a una consulta no coincide completamente con el conjunto de documentos recuperados por el SRI. Debido a la imprecisión con que los modelos de recuperación representan el contenido de los documentos y de las consultas, los mecanismos de recuperación no pueden trabajar con exactitud. Por eso, el sistema recupera muchos documentos que no son relevantes y, además, todos los documentos relevantes tampoco están en la respuesta. Dependiendo de las características del modelo de representación del SRI y de los algoritmos de recuperación que ejecuta, su capacidad de recuperar variará. Por eso, antes de decidirse por un SRI es importante medir y comparar su eficacia.

Para medir la eficacia de las respuestas de un SRI se necesitan dos cosas: una medida de evaluación y un test de referencia. Un test de referencia consiste en un conjunto de documentos, una serie de consultas de prueba y un conjunto de documentos relevantes para cada una de estas consultas. Los documentos que son relevantes para cada consulta han sido escogidos manualmente por una serie de expertos de entre el conjunto de documentos del test. Para medir la eficacia de un sistema debemos aplicar la medida de evaluación y así poder comparar la solución propuesta por nuestro sistema con la solución propuesta por los expertos. En esta sección vamos a estudiar las dos medidas de evaluación de SRI más comunes llamadas memoria y precisión.

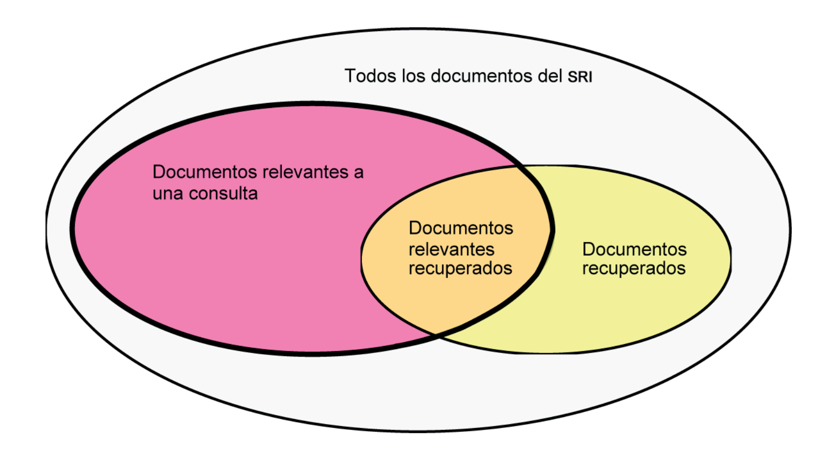


Figura 2.7. Documentos en las respuestas de un SRI

Supongamos que disponemos del conjunto de documentos de un test de referencia almacenado en el SRI que queremos evaluar, y de una consulta de prueba cuya respuesta debe tener un conjunto de documentos relevantes establecidos por un experto, entonces:

La memoria mide el porcentaje de documentos relevantes de la respuesta con respecto al número de documentos relevantes de todo el SRI, es decir, indica la capacidad de recordar del SRI.

$$\text{Memoria} = \frac{\text{Número de documentos relevantes recuperados}}{\text{Número de documentos relevantes a una consulta}}$$

La precisión índica la proporción de documentos relevantes de la respuesta con respecto al número de documentos de la respuesta, es decir mide el nivel de precisión de las respuestas del sistema.

$$\text{Precisión} = \frac{\text{Número de documentos relevantes recuperados}}{\text{Número de documentos recuperados}}$$

Por ejemplo, supongamos que disponemos del conjunto de documentos de un test de referencia almacenado en el SRI que queremos evaluar, y de una consulta de prueba cuya respuesta debe contener un conjunto de 100 documentos relevantes establecidos por un experto. Entonces, si ejecutamos la consulta en el SRI y su respuesta es de 60 documentos relevantes y 180 irrelevantes, tendremos que la memoria del sistema con respecto a la consulta es de 60/100 = 0,6 (60 %) y su precisión de 60/240 = 0,25 (25 %).

Después de ejecutar todas las consultas del test de referencia sobre nuestro SRI podremos resumir los resultados obtenidos en una curva de precisión versus memoria. Esta curva se puede comparar con las curvas obtenidas por otros SRI, y de esta manera comprobar la eficacia de nuestro sistema. En la figura 2.8 se presenta un gráfico que compara la eficacia de dos sistemas SRI A y SRI B.

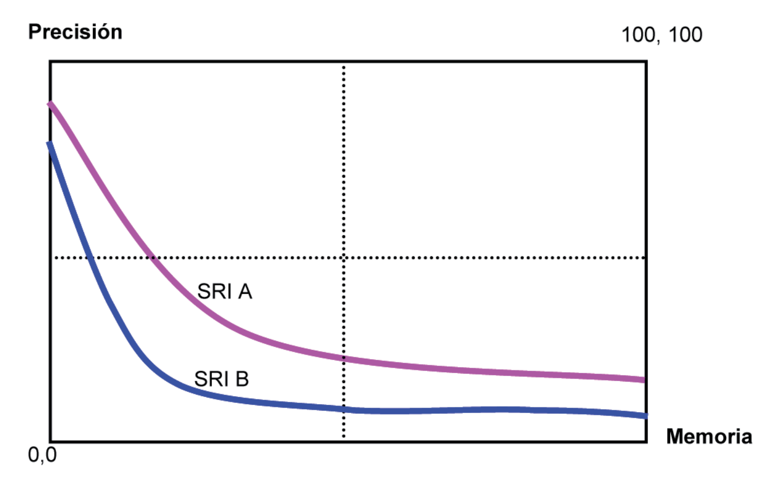


Figura 2.8. Relación entre la memoria y la precisión de un SRI

Como puede observarse, la eficacia del sistema SRI A es siempre comparativamente mejor que la del sistema SRI B. Sin embargo, las dos curvas están siempre bastante lejos de poder obtener una precisión y un memoria del 100 %. Más bien se observa que ambas medidas se contradicen y que una buena memoria produce una baja precisión, y viceversa, una precisión alta produce un memoria muy baja. Este comportamiento es común para todos los modelos y algoritmos de recuperación de información que se han diseñado hasta la fecha. De hecho, con un SRI de propósito general se considera imposible que se pueda llegar a combinar una memoria y una precisión ambos superiores al 60 %.

7. Mecanismos de consulta

Una consulta es la especificación de las necesidades de información de un usuario a través de una interfaz proporcionada por el sistema**.** En su forma más simple, una consulta se compone de unas cuantas palabras clave o frases entre comillas. Si se dispone de mecanismos de búsqueda avanzada, el usuario podrá especificar condiciones más complejas como pueden ser patrones de búsqueda y condiciones de recuperación sobre los metadatos y la estructura de los documentos. En otros sistemas, el usuario es capaz de especificar qué documentos de la respuesta le parecen más relevantes a fin de que el sistema mejore la respuesta buscando otros documentos con características similares. En esta sección se desarrollan los conceptos relacionados con estos mecanismos de búsqueda.

7.1. Palabras clave

Cuando una consulta simplemente se compone de palabras clave, el sistema busca los documentos que contengan dichas palabras. Este mecanismo de consulta es muy popular ya que tiene varias ventajas como que es intuitivo, fácil de usar y que permite ordenar la respuesta por relevancia. En muchas ocasiones además de palabras clave, los SRI permiten especificar entre comillas frases de varias palabras que deben aparecer exactamente igual en los documentos recuperados.

Algunos sistemas tienen mecanismos de búsqueda avanzada que permiten formular consultas indicando unas palabras clave que deben aparecer cercanas, en el mismo orden proporcionado o en cualquier otro. En estas condiciones de proximidad se pueden diferenciar entre dos tipos de cercanía. El primer tipo de cercanía se refiere a que las palabras de la consulta pertenecen a la misma frase del documento original. Por ejemplo, si las palabras clave son 'recuperación' y 'documentos' entonces se pueden encontrar documentos que contengan las frases 'recuperación de documentos', o 'sistemas de recuperación para documentos', o 'los documentos y su recuperación', etc. A pesar de ser muy útil, esta función la proporcionan pocos SRI.

En el segundo tipo de cercanía se proporciona al SRI una secuencia de palabras clave junto con un parámetro que indica el número máximo de palabras que pueden aparecer entre las proporcionadas. Esta función de cercanía es más frecuente que la anterior y a veces el número de palabras de separación es un parámetro pre-establecido por el SRI. Este tipo de consultas permite obtener buenos resultados ya que la mayoría de las veces las palabras que se encuentran cercanas aparecen en la misma frase o en el mismo párrafo lo cual indica que se relacionan semánticamente.

Otra manera de relacionar las palabras clave de una consulta avanzada es por medio de operadores booleanos. Estos operadores se pueden ir anidando formando expresiones complejas. Los operadores más habituales son OR, AND y BUT, donde la consulta (k_1 BUT k_2) recupera todos los documentos que contienen la palabra clave k_1 pero no la k_2. Las consultas booleanas son difíciles de formular sobre todo cuando se utilizan varios operadores.

Como ya hemos visto, el modelo booleano no permite ordenar los documentos del resultado por relevancia, ya que todos ellos cumplen igualmente la expresión formulada (simplemente la hacen cierta). Para evitar este inconveniente de manera que el usuario encuentre más fácil la utilización de expresiones booleanas, algunos SRI relajan su evaluación. Es decir, que si por ejemplo la consulta se compone de varias palabras clave unidas por operadores AND, no se requiere que todas ellas aparezcan en los documentos recuperados, sino que con contener algunas es suficiente. Así, cuantas más palabras clave aparezcan en el documento recuperado mayor será su nivel de relevancia.

7.2. Patrones de búsqueda

Las consultas que incluyen patrones de búsqueda se utilizan para indicar características sintácticas que deben ocurrir en algún fragmento del texto. Es decir, el usuario está interesado en recuperar documentos con unos segmentos textuales que cumplan los patrones especificados. Los tipos de patrones más comunes son:

- **Palabras clave**. Una cadena de caracteres que coincide con una palabra del texto.
- **Prefijos**. Una cadena que se corresponde con el principio de alguna palabra del texto. Por ejemplo, el prefijo 'comput' recupera documentos conteniendo palabras como 'computadora', 'computación', 'cómputo', etc.

- **Sufijos**. Una cadena que se corresponde con el final de alguna palabra del texto. Por ejemplo, el sufijo 'ordenador' recupera documentos conteniendo palabras como 'ordenador', 'miniordenador', 'microordenador', etc.
- **Subcadenas**. Una cadena que aparece en cualquier posición de una palabra del texto. Por ejemplo, la subcadena 'proces' recupera documentos conteniendo palabras como 'procesamiento', 'multiprocesador', 'macroproceso', etc.
- **Rangos**. Considerando el orden lexicográfico de las palabras que coincide con el orden de un diccionario, con este mecanismo se proporcionan dos palabras para recuperar los documentos que contengan cualquiera de las palabras que se localizan en el diccionario entre estas dos inclusive.
- **Errores permitidos**. En este caso se especifica una palabra y un parámetro que indica un número de errores permitidos. De esta manera, se recuperan textos que contengan dicha palabra o cualquier otra que sea similar. Esto sirve para evitar que la búsqueda se vea afectada por errores tipográficos o faltas de ortografía.
- **Expresiones regulares**. Los sistemas más avanzados permiten utilizar expresiones regulares. Una expresión regular es un patrón muy general en cuya especificación se pueden utilizar operadores con cadenas como son los comodines, la opción, la repetición de un número indeterminado de veces, y la concatenación.

7.3. Relevancia de usuario

Cuando un usuario comienza el proceso de búsqueda de información, normalmente requerirá formular varias consultas antes de encontrar los documentos que necesita. Después de observar los documentos que le devuelva una consulta, el usuario reformulará esta para ajustarla mejor a sus necesidades. Probablemente la observación de los documentos recuperados le proporcione pistas de cómo modificar su consulta inicial.

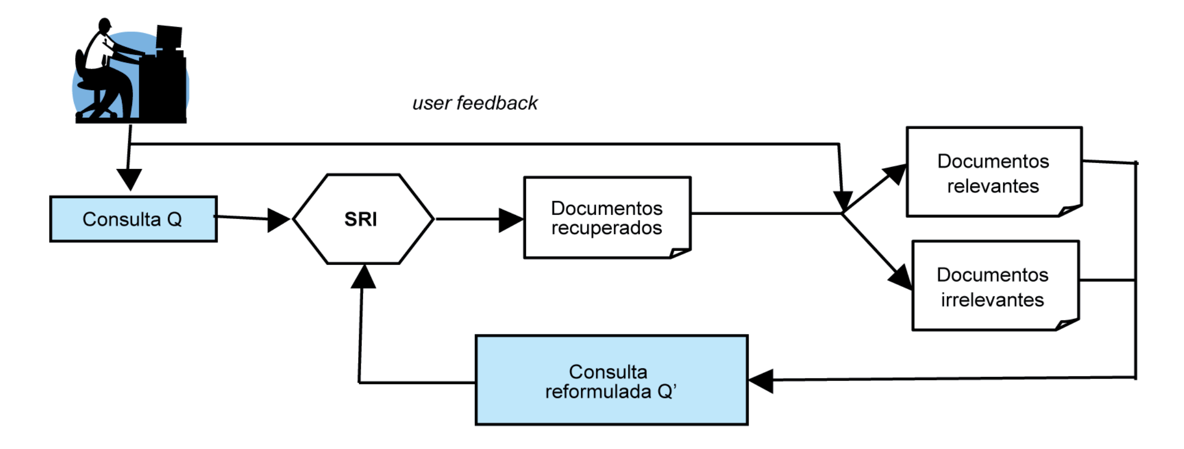


Figura 2.9. La realimentación del usuario sobre la relevancia de los documentos recuperados

Esta labor también puede ser hecha de manera automática por el SRI. La estrategia más frecuente de reformulación de consultas es la llamada realimentación del usuario sobre la relevancia de los documentos recuperados o *user feedback*. En este caso el usuario solamente tiene que indicar qué documentos de la solución le parecen más relevantes y cuáles menos. Entonces el sistema selecciona algunos términos o características de los documentos que el usuario ha encontrado más relevantes para aumentar su importancia en la reformulación de la consulta, y viceversa, también tiene que disminuir la importancia de los términos asociados a los documentos menos relevantes. Una manera de aumentar o disminuir la importancia de un término es ajustando su peso en la consulta. Como se muestra en la figura 2.9, el efecto que se persigue con la reformulación es que el resultado de la nueva consulta tenga más documentos relevantes y menos no relevantes.

7.4. Recuperación de documentos estructurados

Cuando un usuario especifica condiciones de recuperación sobre una colección de documentos estructurados puede elegir entre utilizar los elementos hoja del árbol de composición de los documentos o alguna de sus componentes complejas. Por ejemplo, considerando una colección de documentos similares al de la figura 2.5, no es lo mismo recuperar documentos que contengan la palabra clave 'Dijkstra' en cualquiera de sus componentes, pudiendo coincidir con el nombre de un autor, que recuperar artículos en cuyo título aparezca dicha palabra clave. Además, el usuario puede estar interesado en recuperar los documentos completos o solamente alguna de sus componentes, como puede ser el título de los artículos y sus autores.

Para poder acceder al contenido de los documentos XML, se ha desarrollado el lenguaje estándar *XPath* (XML *Path Language*). Con este lenguaje se utilizan los nombres de las etiquetas para escribir trayectorias que indican cualquier elemento o atributo dentro de un documento XML. Por ejemplo, la expresión SigmodRecord/issues/articles/articlesTuple/authors/author/@AuthorPosition permite localizar en el documento de la figura 2.5 el contenido de los atributos AuthorPosition. El lenguaje *XPath* no solo sirve para acceder al contenido de los elementos almacenados en un documento XML, también sirve para poner condiciones de recuperación sobre los mismos. Por ejemplo, la expresión SigmodRecord/issues/articles/articlesTuple[authors/author]/title permite localizar en el documento de la figura 2.5 el título de los artículos que contengan algún autor dentro de la sección de autores. Como puede observarse, el primer artículo de la figura no cumple esta condición, mientras que el segundo sí. Otros lenguajes de la familia XML que se basan en *XPath* son el lenguaje *XQuery* para recuperar documentos XML y el lenguaje de transformación de documentos XSLT.

8. Almacenamiento de documentos en un SRI

Previamente a ser almacenados en un SRI, los documentos deben ser procesados de diferentes maneras, dependiendo de cada SRI. Su principal cometido es indexar el contenido de los documentos, es decir, procesarlos para extraer los términos de indexación que describen su contenido. Una vez extraídos, estos términos se añaden a un esquema de indexación que permite evaluar las condiciones de la consulta y recuperar los documentos rápidamente. En muchas ocasiones, el proceso de indexación incluye la normalización de los términos de indexación a los conceptos que aparecen en un tesauros. En esta sección vamos a desarrollar mejor los detalles de este proceso.

La mayoría de los SRI que almacenan documentos estructurados también indexan la estructura de los documentos que almacenan. Para ello utilizan técnicas de procesamiento que relacionan el contenido y la estructura del documento, y almacenan esta información codificada en algún esquema de indexación.

Es importante resaltar que una vez han terminado de indexar los documentos, y previamente a almacenarlos en los correspondientes sistemas de ficheros, algunos SRI pueden aplicar técnicas de compresión de datos que permiten ahorrar espacio de almacenamiento y técnicas de encriptación de datos que aumentan la seguridad de los accesos.

8.1. Etapas del proceso de indexación de documentos

A la hora de indexar los documentos hay dos alternativas básicas. Hay sistemas que indexan solamente las palabras que tienen más significado (sustantivos y adjetivos), y otros que indexan absolutamente todas las palabras del texto (*full-text indexing*). La mayoría de los buscadores de Internet siguen una técnica de indexación *full-text* ya que permite un fácil manejo para cualquier tipo de usuario.

Aunque cada SRI tiene su manera de indexar los documentos, puede decirse que un proceso de indexación genérico se basa en las siguientes cinco operaciones de transformación del texto:

1. El análisis léxico del texto tiene como objetivo eliminar los espacios, dígitos, guiones, puntuaciones, mayúsculas y acentos. Para la mayoría de los SRI, todos estos elementos son irrelevantes a la hora de recuperar los documentos, y por lo tanto no deben formar parte del índice ni tampoco deben utilizarse en las consultas de usuario.

2. La eliminación de las palabras de parada o *stopwords* permite filtrar aquellas palabras que no tienen capacidad de discriminación del texto. Esto ocurre con aquellas palabras que pueden aparecen en cualquier documento, independientemente de su tema concreto. Se puede decir que si una palabra

aparece en el 80 % de los documentos entonces es una *stopword*. Se ha comprobado que la eliminación de las *stopwords* permite reducir un 40 % el tamaño de los índices del SRI, con el consiguiente aumento de la velocidad de respuesta.

3. Al eliminar los prefijos y los sufijos de las palabras (*stemming*), se consigue extraer sus raíces gramaticales lo que permite recuperar documentos que contengan cualquier variación de una palabra dada. Por ejemplo ordenar, ordenación, orden, órdenes, etc. El indexar las palabras por sus raíces además de hacer que el tamaño del índice se reduzca, permite indexar los documentos por elementos que se aproximan más a conceptos que a simples palabras. A pesar de que esto pueda parecer una ventaja, en la práctica real no siempre se ha demostrado un beneficio claro de su utilización, por lo que la mayoría de los SRI no la soportan.

4. El proceso de selección de los términos de indexación sirve para determinar qué palabras se van a utilizar para indexar al documento. Normalmente se tratará de nombres y adjetivos ya que contienen mucha más semántica que los verbos y adverbios. Sin embargo, un término de indexación también puede ser una frase nominal como por ejemplo 'grado en ingeniería informática'. Una frase nominal es un grupo de palabras que se utilizan para expresar un concepto.

5. La construcción de categorías de términos en forma de tesauros es un procedimiento que sirve para recuperar los documentos que contengan una palabra o cualquiera de sus sinónimos.

En un SRI concreto todas estas etapas pueden aplicarse en menor o mayor medida, haciendo que el modelo de documento soportado por el sistema pase de un simple *full-text indexing* a un complejo procesamiento basado en términos de indexación con alto nivel semántico. El procesamiento que se haga para indexar los documentos tiene dos efectos contrarios. Por un lado mejora la eficacia del sistema ya que hace que sus respuestas se ajusten mejor a los requerimientos del usuario. Sin embargo, los sistemas que no utilizan una indexación *full-text* son menos intuitivos de utilizar y requieren que los usuarios tengan una preparación previa que les ayude a formular sus consultas y a interpretar los resultados. Es decir, cuanto mejor conozca el usuario la manera en que están indexados los documentos de su SRI, mejor sabrá cómo utilizarlo para recuperar información.

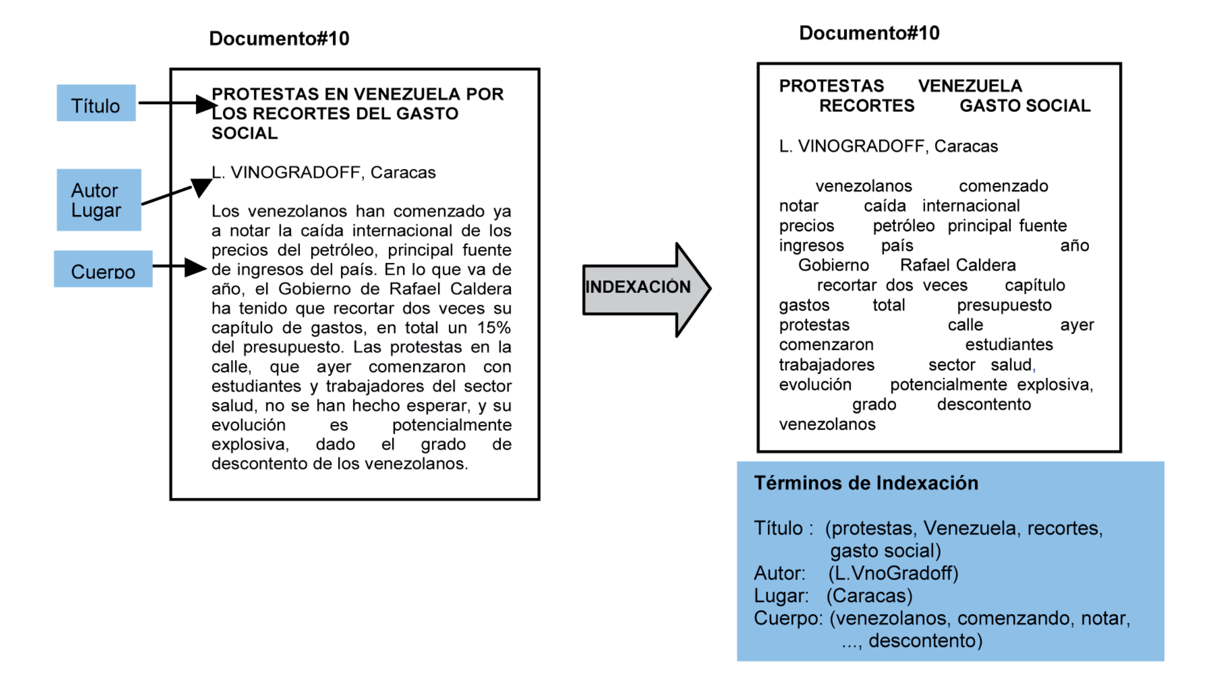


Figura 2.10. Proceso de indexación de un documento ejemplo

Como se observa en el ejemplo de la figura 2.10, cuando el SRI representa la estructura de los documentos, el proceso de indexación debe hacerse separadamente para cada componente del documento original. De esta manera, en la parte izquierda de la figura 2.10 se muestra el texto original después de que se hayan identificado sus componentes lógicas. En la parte derecha de la figura se muestra el texto una vez procesado para su indexación. Como puede observarse, todas las *stopwords* han sido eliminadas y las palabras restantes constituyen los términos de indexación de cada componente del documento.

8.2. Tesauros

En su forma más simple un tesauro es una lista de palabras a las que se asocian otras palabras sinónimas o semánticamente relacionadas. Sin embargo, los tesauros no siempre se organizan como una simple lista de palabras, sino que a veces agrupan los conceptos en categorías y subcategorías, o por otro tipo de relaciones de clasificación y asociación. Algunos tesauros también permiten conocer los diferentes significados o acepciones que una palabra puede tener.

El principal propósito de un tesauro es proporcionar al SRI un vocabulario estándar para indexar y buscar los documentos. Con esta técnica, el proceso de indexación consiste en localizar dentro de los documentos los términos de indexación que aparecen en el tesauro. Cuando se utiliza un tesauro para indexar los documentos, cada término de indexación define un concepto, y puede estar formado por solo una palabra o por una frase nominal. Esto se debe a que hay conceptos que no se pueden decir con una sola palabra, como por ejemplo: ‘Seguridad Social’, ‘fibra óptica’ o ‘Don Quijote de la Mancha’.

Cuando el usuario formula las palabras clave de su consulta, el sistema previamente localiza en el tesauro las entradas correspondientes y después recupera los documentos que incluyan las palabras clave de la consulta o cualquiera de sus sinónimas definidas en el tesauro. A la hora de recuperar información, los tesauros también pueden ayudar a los usuarios a encontrar los términos que mejor definen sus necesidades de información. En el caso de que la respuesta del sistema no se ajuste mucho a sus necesidades por tener pocos o demasiados documentos, el usuario debe buscar términos más generales que le devuelvan más documentos, o términos más específicos que le reduzcan el número de documentos de la respuesta. En ambos casos, un tesauro proporciona una jerarquía de clasificación de palabras clave que pueden ayudar al usuario a encontrar los términos para reformular su consulta. Si el tesauro relaciona los conceptos por relaciones de asociación con diferentes significados, los usuarios también pueden encontrar palabras semánticamente relacionadas que puedan ser incorporadas en su consulta para obtener mejores resultados.

La utilización de tesauros es especialmente recomendable en los SRI de un ámbito concreto. Por ejemplo, en una ciencia como la medicina se puede definir un tesauro que proporcione un vocabulario normalizado para indexar los documentos por conceptos médicos específicos que faciliten su recuperación por profesionales de la salud. Sin embargo, en un SRI de ámbito general como pueden ser aquellos que aparecen en Internet, su utilización es mucho más difícil. Esto se debe a que la definición de un tesauro de ámbito general para un SRI con contenidos de temática variopinta es muy compleja. Además, las consultas pueden ser tantas y provenientes de tantos contextos diferentes, que incluso disponiéndose de un tesauro, su utilidad real es muy limitada.

9. Técnicas de indexación y búsqueda

A la hora de procesar las consultas de un SRI, la opción más simple consiste en recorrer secuencialmente los documentos almacenados buscando las palabras clave especificadas por los usuarios. La técnica de recorrido secuencial se puede utilizar en SRI que no preprocesan los documentos insertados y que por lo tanto no disponen de un esquema de indexación sobre los mismos. Esto suele ocurrir cuando se almacenan documentos muy volátiles (que sufren muchas modificaciones), ya que el coste de mantenimiento del índice puede ser demasiado alto. Esta técnica también puede ser necesaria en SRI que no disponen de suficientes espacio físico para almacenar los documentos junto con sus índices. Los tiempos de respuesta de este método pueden ser suficientemente buenos para SRI con pocos documentos o documentos muy pequeños.

Sin embargo, en la gran mayoría de los casos, se utiliza la opción de construir esquemas de indexación sobre los textos para acelerar las búsquedas. Resulta rentable mantener un índice en el caso de que el tamaño de la colección sea grande y los documentos bastante estáticos (que sufren muy pocas modificaciones). Los SRI con un tamaño intermedio suelen utilizar técnicas de procesamiento de consultas que combinan la utilización de un índice con algún método de búsqueda secuencial.

En el resto de esta sección se explica la técnica de indexación para texto más eficiente y más utilizada, que son los llamados índices invertidos. Previamente se explica un tipo de estructura de datos con forma de árbol que se aplica frecuentemente para construir ficheros invertidos. Finalmente se presentan los ficheros de signaturas que constituyen otro tipo de índice muy utilizado porque permite evaluar las consultas booleanas muy rápidamente.

9.1. *Tries*

Un *trie* es un árbol no binario que se utiliza para implementar diccionarios de términos. En un *trie* se almacena un conjunto indeterminado de cadenas de caracteres que pueden ser recuperadas en un tiempo que es independiente del número de cadenas almacenadas y que es directamente proporcional a la longitud de la cadena recuperada.

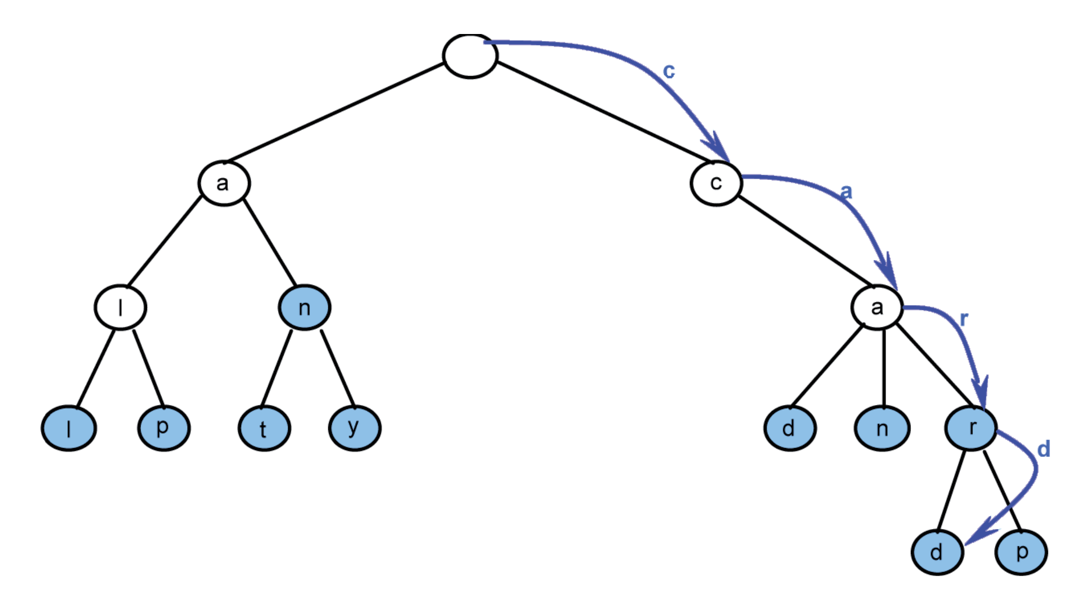


Figura 2.11. Ejemplo de un *trie*

En la figura 2.11 se muestra un ejemplo muy simple de *trie* con palabras en inglés. Cada nodo, excepto el nodo raíz, se asocia con una letra y puede tener tantos nodos hijo como letras haya en el alfabeto. Para diferenciar las cadenas completas de los prefijos se marcan los nodos donde termina alguna palabra (nodos marcados en color oscuro). Para buscar una cadena en el *trie*, se empieza por el nodo raíz y se desciende por las ramas adecuadas a cada carácter. Por ejemplo, en el árbol anterior se muestra la trayectoria seguida para recuperar la cadena '*card*'.

La principal ventaja de los *tries* con respecto a otras estructuras de datos es su tiempo de ejecución, ya que la búsqueda de una clave de longitud m tendrá en el peor de los casos un coste de orden $O(m)$. Además, requieren poco espacio de almacenamiento puesto que las palabras no se almacenan explícitamente, solamente se almacena la estructura de datos que implementa el *trie*.

A la hora de implementar un diccionario de términos, una buena alternativa a los *tries* son las tablas *hash*. Una tabla *hash* es una estructura de datos que asocia claves con valores por lo que normalmente se implementa con un vector. El acceso a los elementos almacenados en el diccionario (vector) se hace por medio de una clave generada a partir de la palabra que se debe buscar. Las tablas *hash* ocupan poco y normalmente caben en memoria principal por lo que proporcionan un acceso muy rápido. Como contrapartida, las claves generadas pueden colisionar (misma clave para dos palabras diferentes), siendo este un problema que complica su utilización. Además, a diferencia de los *tries*, las tablas *hash* no permiten acceder a los términos utilizando prefijos.

9.2. Ficheros invertidos

Un fichero invertido es un esquema de indexación para colecciones de textos que tiene como finalidad aumentar la velocidad de recuperación de los SRI. Como se observa en la figura 2.12, un fichero invertido consta de dos elementos: el diccionario de términos y las listas de ocurrencias. El diccionario de términos es el conjunto de todas las palabras que se encuentran en los textos. Para cada palabra se almacena una lista de ocurrencias. En cada registro de ocurrencias se almacena la posición que ocupa la palabra dentro del documento y un puntero a la dirección exacta de la palabra en el documento. Una posición en el documento se expresa como el número de la palabra dentro del texto (es decir, la i-ésima palabra).

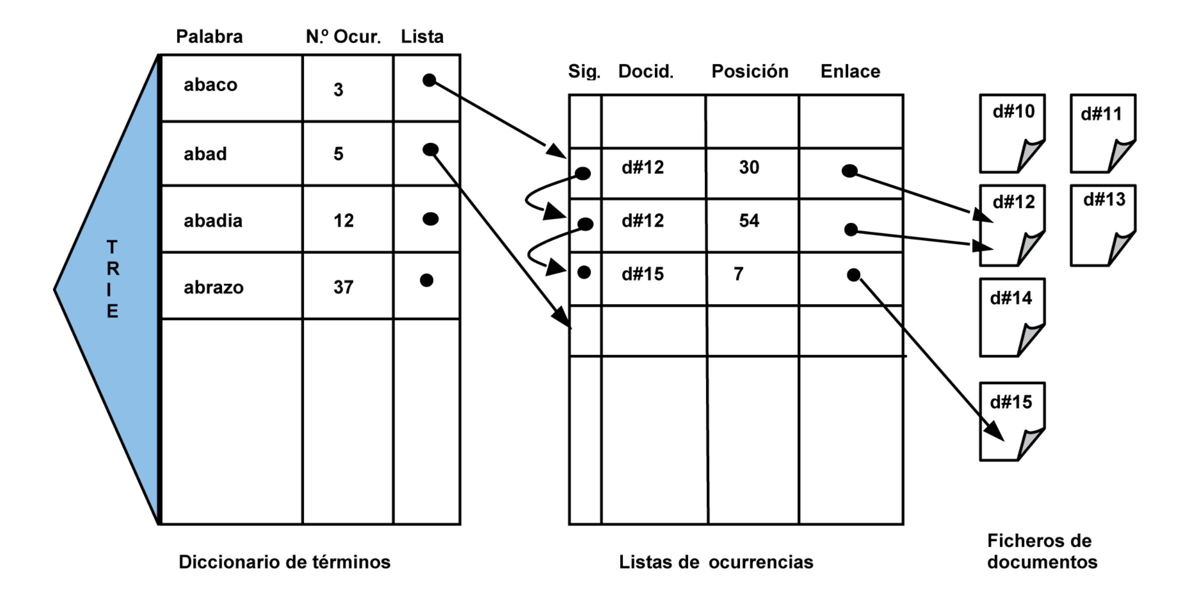


Figura 2.12. Esquema genérico de un fichero invertido

Una de las mejores maneras de construir el diccionario de términos de un fichero invertido es utilizar un *trie*, ya que de esta manera las palabras se localizan muy rápidamente independientemente de cuántas sean. Aunque el diccionario de términos ocupa poco espacio de almacenamiento, las listas de ocurrencias de las palabras suelen ocupar mucho. Esto se debe a que los enlaces apuntan exactamente

a la dirección de memoria donde aparece la palabra dentro del documento. Una técnica para ahorrar espacio con los enlaces al texto consiste en hacer un direccionamiento por bloques. En este caso el texto se divide en bloques y las ocurrencias apuntan al bloque donde se encuentra la palabra, en lugar de su posición exacta. Así, al haber menos bloques, los enlaces que indican las posiciones de los documentos ocupan menos espacio. Para procesar las consultas, primero se utiliza el fichero invertido para localizar los bloques, que luego pueden ser recuperados para buscar las ocurrencias de las palabras de búsqueda.

9.2.1. Búsqueda en un fichero invertido

El algoritmo de búsqueda en un fichero invertido tiene los siguientes pasos principales:

1. Las palabras y los patrones de la consulta son buscados individualmente en el diccionario de términos. Antes de comenzar la búsqueda, tanto las frases como las condiciones de proximidad se dividen en palabras independientes.
2. Se recupera la lista de ocurrencias de todas las palabras encontradas en el diccionario de términos.
3. Se procesan las ocurrencias recuperadas para terminar la evaluación de condiciones de proximidad, frases y operadores booleanos. Si se esta utilizando un direccionamiento por bloques, para evaluar algunas de estas condiciones puede ser necesario procesar todo el texto del bloque.
4. Se recuperan los ficheros de los documentos que cumplen la consulta.

9.2.2. Construcción de un fichero invertido

A la hora de construir un fichero invertido lo mejor es dividirlo en dos ficheros. En uno se almacena las listas de ocurrencias enlazadas por punteros (ver figura 2.12). En el otro fichero se almacena el diccionario de términos. Aquí, junto con cada palabra hay un número que indica cuántas veces ocurre esa palabra en los textos almacenados y un puntero en la posición del fichero de ocurrencias donde comienza su lista de ocurrencias.

El diccionario de términos debe almacenarse en un fichero independiente ya que la mayoría de las veces cabrá en memoria principal con lo que su procesamiento será muy rápido. Para su implementación se pueden utilizar árboles de búsqueda (B-*trees*), tablas *hash* o *tries* como los explicados anteriormente. Tanto las tablas *hash* como los *tries* tienen tiempos de búsqueda O(m), siendo m la longitud de la palabra buscada. Los árboles de búsqueda también son bastante rápidos ya que se recorren con algoritmos cuyo coste es del orden O(log n), siendo n el número de palabras del diccionario. Otra buena alternativa es almacenar las palabras del diccionario de términos en orden lexicográfico dentro de un fichero secuencial ya que así ocupan muy poco. En este caso los tiempos de búsqueda son también buenos ya que con una técnica de búsqueda binaria se pueden hacer algoritmos con un coste O(log n), siendo n el número de palabras del diccionario.

Construir y mantener un diccionario de términos en un fichero invertido resulta barato ya que ocupa poco y se puede crear en tiempo lineal. Supongamos que se almacena en un *trie* y que cada palabra se asocia con una lista de ocurrencias. Para indexar un documento primero se extraen sus términos de indexación y a continuación se busca cada uno de ellos en el *trie*. Si un término no se encuentra en el *trie*, se añade y se crea una nueva lista de ocurrencias con un nuevo registro por cada vez que aparece la nueva palabra en el texto. Si el término de indexación ya se encuentra en el *trie*, entonces se insertan y se enlazan las nuevas ocurrencias al final de la lista de ocurrencias. Una vez esta operación se ha completado para todos los términos de indexación del documento, entonces el *trie* se escribe en disco junto con el nuevo fichero de ocurrencias.

9.3. Ficheros de signaturas

El principal objetivo de este esquema de recuperación de textos es realizar un filtrado inicial que sea muy rápido para descartar la gran mayoría de los documentos almacenados y recuperar un pequeño grupo de ellos sobre los que seguir procesando la consulta. Es decir, ante la falta de precisión del método de filtrado inicial, los textos del grupo recuperado deberán ser procesados uno a uno para determinar cuáles son los que verdaderamente cumplen la consulta. Los documentos que se recuperan y no cumplen las condiciones de la consulta se denominan *false drops*.

Con este método, cada texto almacenado se asocia con una signatura que de alguna manera rápidamente procesable exprese su contenido con suficiente precisión. Todas las signaturas de todos los textos se almacenan secuencialmente en un fichero sobre el que se realiza la primera operación de búsqueda que descarta la mayor parte de los textos almacenados y recupera los que parecen cumplir las condiciones de la consulta. Para detectar los *false drops* es necesario procesar el contenido de los textos recuperados.

El mejor método de crear signaturas es el que se denomina *superimposed coding*. Con este método cada palabra del vocabulario se asocia con una cadena binaria. Para generar la signatura de un texto se hace el OR de todas las signaturas de las palabras que contenga (o solo de los términos que se quieran indexar). En el ejemplo de la figura 2.13 se muestra cómo crear la signatura para el texto 'bases de datos avanzadas'.

PALABRA	SIGNATURA
base	001 000 110 001
dato	010 010 100 001
avanzada	100 010 000 101
SIGNATURA DEL TEXTO	111 010 110 101

Figura 2.13. Ejemplo de creación de signaturas

En este caso las signaturas son de 12 bits, y con 4 bits a 1 por cada palabra. Cuanto mayor sea la longitud de la signatura, menor será la posibilidad de *false drops*, aunque el fichero de signaturas ocupará más y la búsqueda será más lenta. Además, para no derrochar espacio de almacenamiento, lo más deseable es que la signatura compuesta de un texto tenga al menos la mitad de sus bits igual a 1.

Cuando se buscan los textos que contengan una palabra clave, se genera la signatura de la palabra clave, normalmente aplicándole una función *hash*, y se descartan todos los textos que no tengan un bit igual a 1 en las mismas posiciones que la palabra clave. Cuando se buscan varias palabras clave unidas por operadores AND y OR, antes de realizar la búsqueda se puede operar con sus signaturas utilizando los mismos operadores. De esta manera, todas las palabras clave se buscan al mismo tiempo.

Como se muestra en el ejemplo de consulta de la figura 2.14, un *false drop* solo puede ser localizado procesando el contenido de los documentos inicialmente recuperados. La consulta del ejemplo **(A and B)** cuya signatura es 0001 resultado de calcular (0011 **and** 0101) recupera el Texto 1 y el Texto 2, el cual es un *false drop* por no cumplir la condición **B**.

Palabra A	001
Palabra B	010
Signatura del Texto 1	011

Palabra A	001
Palabra C	100
Signatura del Texto 2	101

Palabra A	0011
Palabra B	0101
Signatura de la consulta (A and B)	0001

Figura 2.14. Ejemplo de *false drop*

Este esquema recuperación de textos se ha utilizado ampliamente para implementar SRI con el modelo booleano. Su ventaja es que es muy fácil de implementar y la inserción de nuevos textos es muy rápida. La desventaja principal es que para repositorios grandes, el acceso al fichero de signaturas es muy lento, sobre todo si se hace secuencialmente.

9.4. Indexación de documentos estructurados

Como cualquier otro tipo de documentos, previamente a ser almacenados en un SRI, los documentos estructurados deben ser procesados e indexados de manera que las condiciones de recuperación sobre su contenido y estructura puedan ser ejecutadas de manera eficiente. En general, las técnicas de indexación que se emplean son las mismas que hemos visto anteriormente pero modificadas para poder ser utilizadas en colecciones de documentos estructurados con un formato procesable, ya sea XNL u otro similar. En esta sección se presentan muy brevemente las principales técnicas de indexación para documentos estructurados.

El método de indexación más básico consiste en procesar el contenido de los elementos hoja para indexar su contenido. En este tipo de índice, además de indicarse

la posición del término dentro del documento, también se indica el elemento hoja al que pertenece. De esta manera, evaluar una condición de recuperación sobre el contenido de un documento o de uno de sus elementos hoja es una tarea fácil. Sin embargo, la ejecución de condiciones de recuperación sobre componentes complejas del documento es más costosa. Esto se debe a que es necesario identificar los elementos hoja que forman parte de las componentes complejas antes de evaluar las condiciones de recuperación sobre sus contenidos. Estos contenidos aparecerán repartidos por las hojas del subárbol cuya raíz es la componente compleja.

Para facilitar esta labor, algunos sistemas permiten crear índices específicos para un elemento concreto dentro de la estructura del documento. Es decir, mediante la especificación de una expresión *XPath* se localiza un elemento complejo dentro de la estructura de los documentos y se crea un índice sobre sus contenidos (los contenidos de las hojas del subárbol cuya raíz es la componente compleja). Lógicamente, el procesamiento del documento previo a su almacenamiento se complica y, si se indexan muchas componentes de esta manera, los índices resultantes pueden ser muy redundantes, ya que el mismo contenido puede estar indexado en los índices de las diferentes componentes complejas que lo contienen. En general, se recomienda utilizar este tipo de índice para aquellas componentes complejas que van a ser frecuentemente utilizadas por los usuarios en sus consultas. Solo en estos casos su coste de ejecución y almacenamiento queda justificado por sus beneficios.

Otros tipos de índices más sencillos son los que sirven para evaluar condiciones de recuperación sobre un elemento o atributo concreto independientemente de su posición dentro de la estructura del documento. De esta manera se puede indexar información muy útil como son los títulos, los autores o las fechas de publicación de los documentos almacenados.

Una vez se haya completado la ejecución de las condiciones de recuperación de las consultas a través de los índices disponibles, se obtendrá una colección de documentos que deberán ser procesados uno a uno a fin de poder ejecutar el resto de condiciones de las consultas. Finalmente, para los documentos que cumplan todas las condiciones de recuperación habrá que devolver al usuario todo su contenido o solamente la subsección especificada en la consulta. Cada uno de estos elementos deberá ir acompañado por un parámetro que proporcione al usuario un grado de relevancia del documento con respecto a la consulta. Dado que las condiciones de recuperación pueden implicar a varias componentes de la estructura del documento, el cálculo del grado de relevancia de un documento a una consulta es más complicado que con los modelos de recuperación clásicos.

Bibliografía

Baeza Yates, R., Ribeiro Neto, B. (1999): *Modern Information Retrieval*, Addison-Wesley Longman.

Baeza Yates, R., Ribeiro Neto, B. (2011): *Modern Information Retrieval: The Concepts and Technology behind Search,* ACM Press Books, Addison-Wesley.

CACHEDA SEIJO, F., FERNÁNDEZ LUNA, J. M., HUETE GUADIX, J. F. (2011): *Recuperación de Información: Un enfoque práctico y multidisciplinar*, Ra-Ma.
CHAUDHRI, A., ZICARI, R. (2003): *XML Data Management,* Addison Wesley.
ELMASRI, R., NAVATHE, S. (2011): *Fundamentals of Database Systems,* (6.ª edición), Addison-Wesley Longman.
LEVINGSTONE, D. (2002): *Guia Esencial XML,* Pearson Alambra.
RAY, E. (2003): *Learning XML,* O'Reilly.
SALTON, G., MCGILL, M. J. (1983): *Introduction to Modern Information Retrieval*, McGraw-Hill.

Ejercicios

Ejercicio 1

Westlaw, en España más conocido como Aranzadi, es un proveedor de sistemas de recuperación de información legislativa que provee una interfaz de consulta basada en un lenguaje de tipo booleano con los siguientes operadores:

- El operador & equivale a AND: 'aeropuerto & Valencia' recupera documentos que contengan la palabra aeropuerto y la palabra Valencia.

- El espacio en blanco equivale a OR: la consulta 'aeropuerto Valencia' recupera documentos que contengan la palabra aeropuerto o la palabra Valencia, o ambas.

- El operador ! sustituye a cualquier sufijo: por ejemplo, 'aerop!' se expandiría tanto a 'aeropuerto' como a 'aeroportuario'.

- El comando /s busca palabras que ocurren en la misma frase ('aeropuerto /s Valencia').

- El comando /p busca palabras que ocurren en el mismo párrafo ('aeropuerto /p Valencia').

- El comando /k busca palabras que ocurren a menos de k palabras de distancia: 'aeropuerto /2 Valencia' devolvería un documento que contuviera 'aeropuerto de Valencia', pero no 'aeropuerto recientemente ampliado en Valencia'.

1.1. ¿Cómo crees que afectará cada uno de los operadores anteriores a la precisión de las consultas? ¿Y a la memoria? Ayúdate de ejemplos que ilustren tu respuesta.

Ejemplo de respuesta: Considera el operador &. Al utilizarlo en una consulta, se restringirán los documentos devueltos a aquellos que contengan las dos palabras. Esto eliminará documentos del conjunto de resultados disminuyendo la memoria, pero en general los documentos devueltos serán más

relevantes aumentando la precisión. Por ejemplo, para encontrar documentos que traten del aeropuerto de Valencia, la consulta 'aeropuerto & Valencia' será más precisa que 'aeropuerto' o 'aeropuerto Valencia' (con un OR implícito), pero tendrá menos memoria ya que algunos documentos relevantes no se mostrarán, por ejemplo los que contienen la frase 'aeropuerto valenciano'.

1.2. Diseña un fichero invertido que se pueda utilizar para implementar el lenguaje de consulta descrito arriba.

1.3. Muestra paso a paso cómo se ejecutaría la consulta 'aerop! /2 valenc! /s situación' usando el fichero invertido que acabas de diseñar.

Ejercicio 2

El sistema de código libre Lucene proporciona muchas funciones para la implementación de un SRI. Contesta a las siguientes cuestiones:

2.1. Lucene permite varios tipos de comodines en las palabras, tanto de una sola letra (el comando 'te?t' reconoce tanto 'test' como 'text') como cualquier subcadena (el comando 'te*x' reconocería 'tex', 'telex', etc.). ¿Es posible implementar el comando '?' mediante el *trie* básico que hemos visto en clase? ¿Y el comodín '*'? Indica unas pautas de cómo hacerlo.

2.2. Lucene soporta un operador AND (por ejemplo 'aeropuerto AND Valencia') y también **+**, que fuerza a que la siguiente palabra esté en el documento (por ejemplo, '+aeropuerto +Valencia'). En todos los modelos de recuperación que hemos visto estas dos consultas no son equivalentes. ¿Por qué crees que es así?

Ejercicio 3

El paquete *tm* (*text mining*) es una librería de código libre para procesamiento de texto disponible para el lenguaje de programación R que contiene las siguientes funciones para el procesamiento de términos de indexación:

- Eliminar espacios en blanco, puntuación o números.
- Convertir todo a minúsculas o mayúsculas.
- *Stemming* (extracción de raíces gramaticales).
- Eliminar *stopwords* (incluye listas en varios idiomas).

3.1. Cada una de estas cuatro opciones tiene efectos sobre la memoria y la precisión del SRI. Explica con ejemplos qué ocurre en cada caso.

3.2. ¿Se te ocurre algún ejemplo de aplicación en la que para extraer los términos de indexación del texto fuera importante…

- … no eliminar los signos de puntuación?
- … no eliminar los números?
- … no normalizar las palabras a mayúsculas/minúsculas?

Da un ejemplo de cada caso.

Ejercicio 4

La biblioteca de la UJI proporciona una interfaz de consulta sencilla para su SRI (http://cataleg.uji.es/). Experimenta un poco con él y, realizando consultas de prueba, averigua sus características:

- ¿Qué campos de metadatos están presentes?
- ¿Las *stopwords* se consideran términos de indexación?
- ¿Se realiza procesamiento de terminaciones lingüísticas?
- ¿Se distinguen mayúsculas/minúsculas? ¿Y acentos?
- ¿Se indexan los números, códigos especiales, etc.?

a	acá	ahí	ajena	ajenas	ajeno	ajenos	al
algo	algún	alguna	algunas	alguno	algunos	allá	allí
ambos	ante	antes	aquel	aquella	aquellas	aquello	aquellos
aquí	arriba	así	atrás	aun	aunque	bajo	bastante
bien	cabe	cada	casi	cierta	ciertas	cierto	ciertos
como	cómo	con	conmigo	conseguimos	conseguir	consigo	consigue
consiguen	consigues	contigo	contra	cual	cuales	cualquier	cualquiera
cualquiera	cuan	cuán	cuando	cuanta	cuánta	cuantas	cuántas
cuanto	cuánto	cuantos	cuántos	de	dejar	del	demás
demás	demasiada	demasiadas	demasiado	demasiados	dentro	desde	donde
dos	el	él	ella	ellas	ello	ellos	empleáis
emplean	emplear	empleas	empleo	en	encima	entonces	entre
era	éramos	eran	eras	eres	es	esa	esas
ese	eso	esos	esta	estaba	estado	estáis	estamos
están	estar	estas	este	esto	estos	estoy	etc
fin	fue	fueron	fui	fuimos	ha	hace	hacéis
hacemos	hacen	hacer	haces	hacia	hago	hasta	incluso
intenta	intentáis	intentamos	intentan	intentar	intentas	intento	ir
jamás	junto	juntos	la	largo	las	lo	los
mas	más	me	menos	mi	mía	mías	mientras
mío	míos	mis	misma	mismas	mismo	mismos	modo

mucha	muchas	muchísima	muchísimas	muchísimo	muchísimos	mucho	muchos
muy	nada	ni	ningún	ninguna	ningunas	ninguno	ningunos
no	nos	nosotras	nosotros	nuestra	nuestras	nuestro	nuestros
nunca	os	otra	otras	otro	otros	para	parecer
pero	poca	pocas	poco	pocos	podéis	podemos	poder
podría	podríais	podríamos	podrían	podrías	por	por qué	porque
primero	primero	puede	pueden	puedo	pues	que	qué
querer	quien	quién	quienes	quienesquiera	quienquiera	quizá	quizás
sabe	sabéis	sabemos	saben	saber	sabes	se	según
ser	si	sí	siempre	siendo	sin	sino	so
sobre	sois	solamente	solo	somos	soy	sr	sra
sres	sta	su	sus	suya	suyas	suyo	suyos
tal	tales	también	tampoco	tan	tanta	tantas	tanto
tantos	te	tenéis	tenemos	tener	tengo	ti	tiempo
tiene	tienen	toda	todas	todo	todos	tomar	trabaja
trabajáis	trabajamos	trabajan	trabajar	trabajas	trabajo	tras	tú
tu	tus	tuya	tuyo	tuyos	ultimo	un	una
unas	uno	unos	usa	usáis	usamos	usan	usar
usas	uso	usted	ustedes	va	vais	valor	vamos
van	varias	varios	vaya	verdad	verdadera	vosotras	vosotros
voy	vuestra	vuestras	vuestro	vuestros	y	ya	yo

Figura 2.15. Lista de *stopwords* en español

Ejercicio 5

Considera la lista de *stopwords* de la figura 2.15 y el siguiente texto extraído del SRI de una agencia de noticias:

> **Un nuevo frente francés en África**
>
> *Por Eduardo Martínez, 5 de abril de 2011*
>
> «Francia está en guerra», anuncia *Libération* en su portada tras la intervención en Costa de Marfil «a petición de la ONU» de la fuerza francesa Licorne (con 1.650 soldados) junto a los cascos azules de la ONUCI, la misión desplegada de Naciones Unidas.

5.1. Busca frases nominales significativas. Recuerda que las frases nominales son expresiones en las que dos o más nombres se unen para designar o nombrar algo nuevo. He aquí algunos ejemplos: camión de bomberos, fin de semana, certificado de notas, etc.

5.2. Aplica las etapas del proceso de indexación de documentos de la sección 6.5 eliminando las *stopwords*, considerando solo las raíces gramaticales y las frases nominales que consideres adecuados.

5.3. ¿Qué campos de metadatos extraerías?

Ejercicio 6

Considera la lista de stopwords de la figura 2.15 y los siguientes documentos extraídos del DOGV:

> ***Documento 1**:*
>
> Conselleria de Justicia y Administraciones Públicas. RESOLUCIÓN de 17 de abril de 2008, del conseller de Justicia y Administraciones Públicas, por la que se publica la relación de aspirantes que han superado las pruebas selectivas de acceso al grupo A, sector administración especial, psicólogos, turno de promoción interna. Convocatoria 68/04. [2008/4918]

> ***Documento 2:***
>
> Conselleria de Justicia y Administraciones Públicas. ORDEN de 21 de abril de 2008, del conseller de Justicia y Administraciones Públicas, por la que se nombra funcionarios de carrera a quienes superaron las pruebas selectivas de acceso al grupo B, sector administración especial, informáticos, convocatorias números 38/2004, 39/2004 y 40/2004. [2008/5021]

> ***Documento 3:***
>
> Conselleria de Bienestar Social Licitación del expediente núm. NMY08/02-2/62. Servicio consistente en la grabación informática de los expedientes de solicitudes del Programa 'Vacaciones Sociales para Personas Mayores de la Comunidad Valenciana 2009'. [2008/4926]

6.1. Busca frases nominales significativas. ¿Cuáles has encontrado?

6.2. Busca al menos cinco términos de indexación para cada documento. Hazlo primero asumiendo indexación de texto completo eliminando las *stopwords* y después considerando solo las raíces gramaticales y las frases nominales que creas adecuados.

6.3. Consideremos una matriz de términos/documentos como una matriz booleana de dos dimensiones que para cada posición (d, t) indica si el documento d contiene al término de indexación denotado por t. Construye una matriz términos/documentos con tres filas para los documentos anteriores y con una columna para cada uno de los términos de indexación que hayas encontrado en el ejercicio anterior.

6.4. Crea tres consultas booleanas que incluyan los operadores AND, OR y NOT, y usa la matriz de términos/documentos para calcular su resultado.

Ejercicio 7

Considera la siguiente matriz de términos/documentos:

	Doc. 1	Doc. 2	Doc. 3	Doc. 4	Doc. 5
pizza	1	1			1
filete		1	1	1	1
dorada	1		1		
agua		1	1		1
vino	1			1	

Calcula el resultado de las siguientes consultas booleanas:

- pizza
- pizza AND filete
- pizza OR filete
- NOT agua

Ejercicio 8

Considera los siguientes documentos con sus términos de indexación:

Doc. 1 = {harina, harina, huevos, yogur, yogur}

Doc. 2 = {huevos, azúcar, leche, leche}

Doc. 3 = {azúcar, azúcar, leche, harina, harina, harina}

Doc. 4 = {yogur, yogur, huevos, huevos}

Doc. 5 = {huevos, leche, leche, harina}

Construye su matriz de términos/documentos con pesos calculados mediante *tf.idf*.

Ejercicio 9

Considera la siguiente matriz de términos/documentos con pesos calculados mediante *tf.idf*:

	Doc. 1	Doc. 2	Doc. 3	Doc. 4	Doc. 5
pizza	0,25	0,25			0,25
filete		1	0,25	0,25	0,25
dorada	1		2		
agua		0,25	0,25		0,25
vino	0,25			0,25	

Calcula el resultado de las siguientes consultas vectoriales que están en formato término/peso y utiliza para ello la medida del coseno:

- pizza/1
- pizza/1, filete/1
- pizza/2, filete/1

Ejercicio 10

Considera una base de datos con diez documentos, $\{d_1, d_2, \ldots, d_{10}\}$, y un conjunto de consultas de prueba $\{q_1, q_2, q_3\}$. A través de una serie de pruebas experimentales hemos determinado que el conjunto correcto de respuestas para cada consulta es:

- Para q_1: $\{d_3, d_4, d_5, d_8\}$
- Para q_2: $\{d_1\}$
- Para q_3: $\{d_5, d_6\}$

Aplicando nuestro nuevo SRI sobre este conjunto de consultas de prueba, los resultados son los siguientes:

- Para q_1: $\{d_3, d_8\}$
- Para q_2: $\{d_1, d_2, d_5\}$
- Para q_3: $\{d_5, d_6, d_{10}\}$

Calcula la precisión y la memoria obtenida por nuestro nuevo SRI para cada consulta.

Ejercicio 11

A partir de los tres documentos del ejercicio 6:

10.1. Busca al menos cinco términos de indexación que sean simples, ni frases nominales ni raíces gramaticales, y que aparezcan en más de uno de los documentos. Crea un *trie* a partir de ellos.

10.2. ¿Cómo usarías el *trie* para evaluar consultas con patrones como consell*?

10.3. Crea un fichero invertido a partir de los términos de indexación encontrados.

10.4. ¿Cómo evaluarías una consulta de proximidad usando el fichero invertido?

Por ejemplo:

a. 'administración especial'
b. 'funcionario NEAR/2 carrera'

CAPÍTULO 3

Bases de datos distribuidas e integración de información distribuida

1. Introducción

En un sistema de base de datos centralizado todos los componentes del sistema residen en un único ordenador denominado servidor de base de datos. Sin embargo, muchas aplicaciones de bases de datos tienen una naturaleza distribuida. Por ejemplo, una compañía de transportes puede tener centros de logística en varias ciudades, o un banco puede tener varios centros de procesamiento de operaciones en uno o varios países. Estas aplicaciones requieren bases de datos distribuidas, conectadas a través de una red de comunicaciones y con servidores de datos en cada sede de la compañía. De esta manera, los usuarios locales pueden tener acceso directo a los datos que están en su servidor local pero, como usuarios globales, también puedan acceder a los datos almacenados en otros servidores.

Después de la expansión de Internet, las bases de datos distribuidas tradicionales han evolucionado en sistemas más abiertos pero que igualmente necesitan compartir datos y que además, no necesariamente se encuentran almacenados en una base de datos. Hoy en día, existen infinidad de aplicaciones que requieren integrar datos de naturaleza y formato muy diferente y que además se encuentran distribuidos, como pueden ser los sistemas gestores de noticias, las bibliotecas digitales, las aplicaciones de comercio electrónico o los sistemas de información médica. Como consecuencia han surgido un gran número de arquitecturas y tecnologías cuyo objetivo es ayudar a desarrollar aplicaciones que requieran la integración de fuentes de datos heterogéneos. El rango de posibles aplicaciones es enorme y cada una tiene sus propios requisitos, por lo que debe ser independientemente analizada con el fin de poder determinar cuál es la mejor solución a los problemas de integración de datos que conlleve.

1.1. Objetivos de aprendizaje

Este capítulo es una introducción a las cuestiones fundamentales de las bases de datos distribuidas y al problema de la integración de información. Los objetivos de aprendizaje de este capítulo se enumeran a continuación:

a) Definir qué es una base de datos distribuida, sus principales características y qué las diferencia de las bases de datos centralizadas.
b) Conocer las etapas del diseño de bases de datos distribuidas, qué es un esquema de fragmentación y réplica.
c) Explicar la problemática que conlleva la ejecución de consultas en bases de datos distribuidas.
d) Enumerar las etapas del procesamiento de consultas en bases de datos distribuidas.
e) Plantear soluciones posibles al problema de propagación de actualizaciones en bases de datos distribuidas.
f) Identificar aplicaciones que requieren integrar información distribuida.
g) Explicar en qué consiste el proceso de integrar información distribuida.
h) Conocer las principales arquitecturas que se han diseñado para desarrollar aplicaciones y sistemas de información que requieran integrar información distribuida.

2. Definición de bases de datos distribuidas

En un sistema de base de datos centralizado todos los componentes del sistema residen en un único ordenador que incluye los datos, el software del sistema de gestión de bases de datos (SGBD), las aplicaciones y los dispositivos de almacenamiento secundario. Este ordenador se denomina servidor de bases de datos y aunque puede ser accedido en modo remoto desde muchas terminales cliente, los datos y el SGBD siempre se localizan en un único ordenador servidor. Sin embargo, tal y como muestra la figura 3.1, los datos de una base de datos distribuida se reparten por varios servidores. Cada uno de estos servidores consiste en un ordenador donde se ejecuta un SGBD con funcionalidades para bases de datos distribuidas y se almacenan los datos que más necesitan los usuarios y las aplicaciones a nivel local. Además, cada servidor está dispuesto a atender peticiones de datos provenientes de servidores remotos y viceversa, a realizar peticiones de datos que se almacenan en nodos remotos. Esta capacidad de compartir datos caracteriza a los sistemas distribuidos.

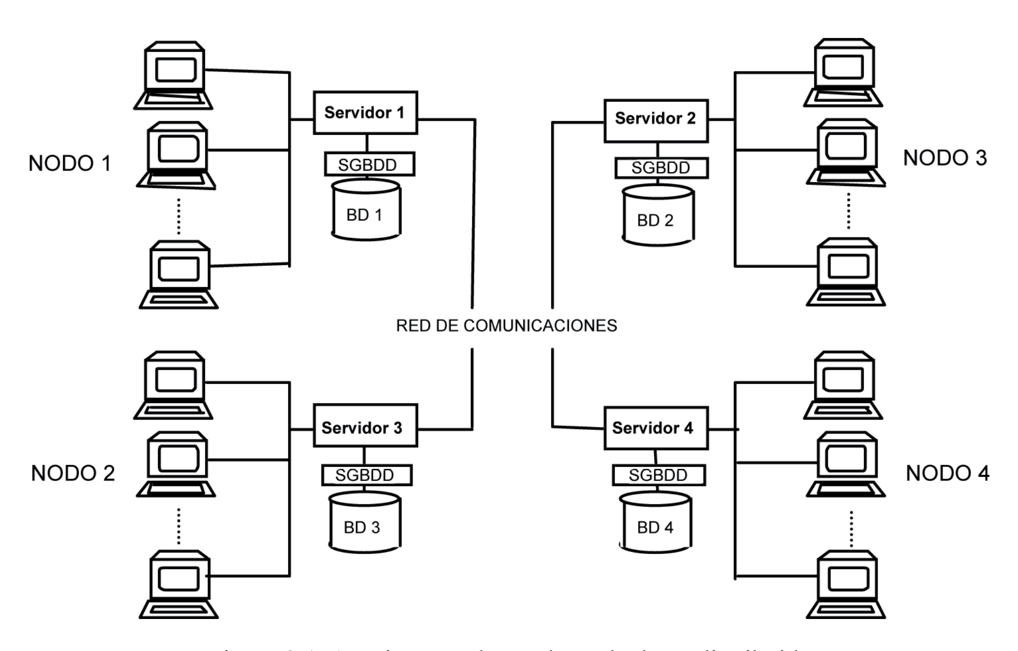


Figura 3.1. Arquitectura de una base de datos distribuida

Un *sistema de bases de datos distribuidas* (SBDD) se define como una colección de múltiples bases de datos que se distribuyen por una red de comunicaciones y que están relacionadas de manera lógica, formando una única base de datos con un solo esquema lógico. Un *sistema de gestión de bases de datos distribuidas* (SGBDD) se define como el sistema software que permite la gestión de bases de datos distribuidas y hace la distribución de datos transparente para los usuarios.

Los servidores de una base de datos distribuida suelen denominarse *nodos* y pueden estar físicamente cercanos (mismo edificio o grupo de edificios) y conectados a través de una red de área local, o pueden estar distribuidos geográficamente a grandes distancias y conectados a través de una red de larga distancia. La red de comunicaciones también puede tener diferentes topologías, lo cual va a influir en

el rendimiento final del sistema. Normalmente, es imprescindible que todos los nodos se puedan conectar entre sí, directamente o a través de otro nodo.

A pesar de que son sistemas muy complejos, un SBDD tiene muchas ventajas con respecto a los sistemas centralizados. La capacidad de almacenamiento de un sistema distribuido es mucho mayor que la de los sistemas centralizados, además su capacidad de crecimiento es también mayor, ya que siempre es más fácil añadir un nuevo nodo que reemplazar un sistema centralizado por otro de mayor capacidad. El número de usuarios y aplicaciones que puede soportar un SBDD es también mucho mayor que en un sistema centralizado.

Otra de las ventajas de los SBDD es que son más fiables porque aunque un nodo o conexión falle, es posible seguir trabajando en otros nodos. Los datos también están más disponibles porque se almacenan varias copias en los nodos donde más se necesitan. Frente a un gran sistema centralizado que almacene todos los datos del sistema y ejecute todas las consultas de sus usuarios, un SBDD puede ser mucho más eficiente si los datos se localizan en donde se utilizan con mayor frecuencia. Hay que tener en cuenta que una consulta sobre una base de datos pequeña siempre es más rápida, y que tener los datos que se necesitan a nivel local elimina el costo de trasferirlos.

3. Acceso a los datos de una base de datos distribuida

Los usuarios de un SBDD deben poder acceder a los datos de la misma manera en que acceden a los de una base de datos centralizada. Tal y como se ilustra en la figura 3.2, cada base de datos cuenta con su propio esquema de datos local que proporciona acceso a sus datos locales. Sin embargo, la mayoría de los usuarios de un SBDD acceden al sistema por un esquema de datos global que se define a partir de los esquemas locales de todas las bases de datos que lo componen, y por el que tienen acceso a todos los datos del SBDD.

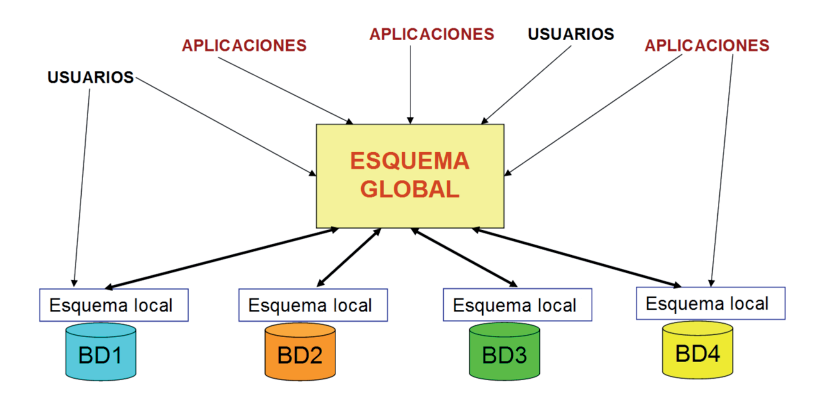


Figura 3.2. Arquitectura de una base de datos distribuida

La *transparencia de localización de los datos* es una propiedad elemental de los SBDD que indica que al formular sus consultas los usuarios no deben especificar

en qué nodos están físicamente almacenados los datos que se necesitan, sino que de cara al usuario las consultas deben poder ser formuladas como si los datos no estuvieran almacenados en diferentes nodos. Es decir, el usuario que formula una consulta contra el esquema global de un SBDD debe hacerlo de la misma manera que lo haría si el sistema fuera centralizado.

Una consulta a un SBDD puede requerir el procesamiento de datos que se almacenan distribuidos por varios nodos. Su ejecución puede causar que los datos sean transportados al nodo local para ser procesados, o alternativamente, que tras su procesamiento remoto los resultados sean enviados desde el nodo remoto a través de la red de comunicaciones. Por esta razón, la transformación y optimización de consultas previa a su ejecución es muy importante en el caso de SBDD. Si en una consulta se da que hay varios nodos implicados y los mismos datos se almacenan en varios de esos nodos, habrá diferentes alternativas para trasladar los datos por la red y procesar la consulta. Dado que independientemente de la red de comunicaciones que se utilice, los tiempos de transferencia de los datos son varios órdenes de magnitud superiores a sus tiempos de procesamiento, ante una consulta es crucial que el SGBDD encuentre una estrategia eficiente para minimizar el tiempo de transferencia de datos.

3.1. El papel del diccionario de datos

Toda la información referente al esquema global de un SBDD se almacena en su diccionario de datos incluyendo además, y entre otras muchas cosas, información acerca de los nodos en donde se puede encontrar cada dato. Antes de ejecutar una consulta distribuida, el procesador del SGBDD consultará el diccionario de datos para decidir a qué nodos debe solicitar los datos. Además de información acerca de los datos que se almacenan en cada nodo, en el diccionario de datos también se dispone de información estadística acerca de los tiempos de respuesta de los nodos y la red de comunicaciones. Toda esta información es utilizada por el procesador de consultas para diseñar una estrategia de evaluación adaptado a cada consulta.

El diccionario de datos de un SBDD se puede almacenar en un solo nodo o se pueden almacenar varias copias en diversos nodos, con lo que su disponibilidad aumentará. El problema de almacenar varias copias es que las operaciones de actualización que haya que ejecutar tardarán más tiempo y serán más complicadas porque deben propagarse a todas las copias al mismo tiempo. Sin embargo, trabajar con una sola copia del diccionario de datos supone un cuello de botella para el SBDD dado el enorme número de consultas que tiene que procesar.

4. Características de los sistemas de bases de datos distribuidas

En esta sección se explican las características principales que puede tener un SBDD, y al mismo tiempo los vamos a clasificar de acuerdo a tres dimensiones: autonomía, heterogeneidad y distribución de datos.

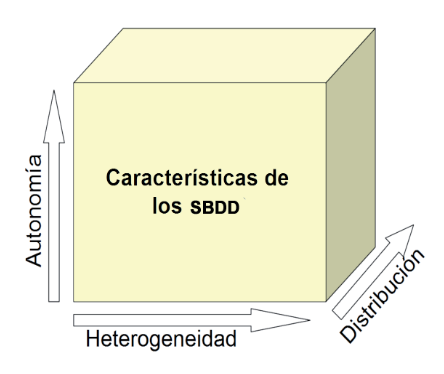


Figura 3.3. Clasificación y características de los SBDD

Como ilustra la figura 3.3, estas tres dimensiones se corresponden con tres criterios de clasificación independientes entre sí, lo cual significa que existe un amplio rango de posibles SBDD con diferentes características y adaptados para resolver problemas de muy diversa índole. En esta sección se desarrolla el significado de cada una de estas dimensiones y las características de los principales tipos de SBDD.

4.1. Autonomía local de los nodos

Un nivel de autonomía alto permite que todas las decisiones que afectan a un nodo se decidan en ese nodo y que no dependa de otros para ejecutar sus propias operaciones. Los SBDD con mayor nivel de autonomía son los *sistemas federados*. A diferencia de los sistemas federados, los nodos de los *sistemas distribuidos totalmente integrados* presentan un nivel de autonomía muy bajo, ya que no pueden ejecutar muchas de sus operaciones sin antes considerar su efecto sobre el SBDD.

Cada base de datos que participa en una federación tiene sus propios usuarios locales a los que debe proporcionar el mismo acceso a los datos que si no formara parte de la federación. Es decir, los usuarios locales pueden acceder a la base de datos local a través de su esquema local, el cual se exporta total o parcialmente a la federación (figura 3.4). Los usuarios del SBDD global se gestionan por separado y acceden al sistema a través del esquema de datos federado, que es la composición de los diferentes esquemas exportados por cada nodo de la federación. Se permite que cada nodo modifique su esquema local independientemente, siempre y cuando el esquema que exportan no se vea alterado. Es posible definir vistas tanto sobre el esquema federado global, como sobre cada esquema local.

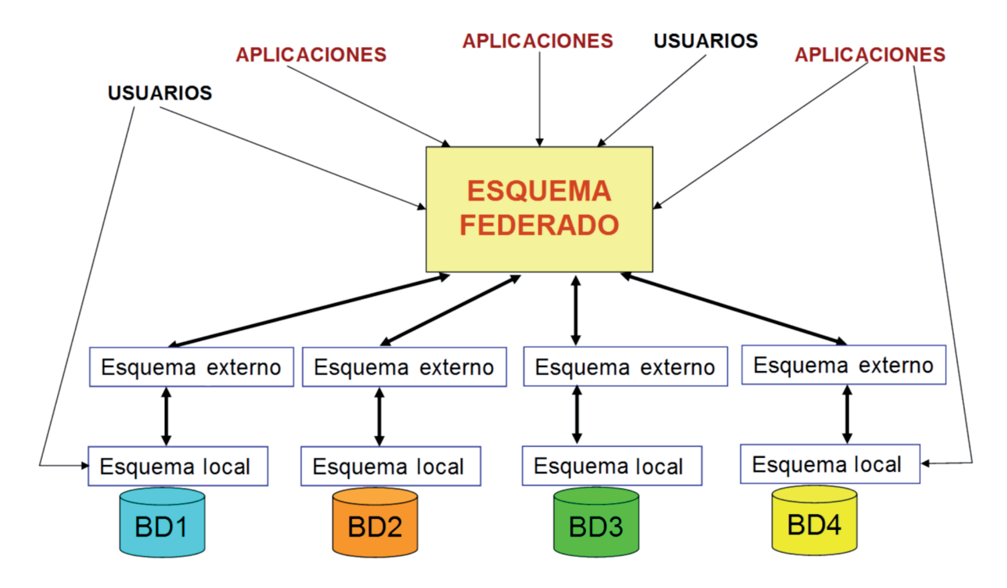


Figura 3.4. SBDD federado

Por el contrario, los *sistemas integrados* (figura 3.5) proporcionan a los usuarios un único esquema de datos, el cual integra totalmente los esquemas locales de cada nodo. Las bases de datos locales no tienen autonomía para tener sus propios usuarios, sino que la única manera de acceder a los datos es a través del esquema integrado global de la base de datos distribuida. Tampoco se permite que los esquemas locales sean modificados independientemente, ya que cualquier modificación a un esquema local afectaría al global. Sobre el esquema integrado global se pueden definir vistas de usuario.

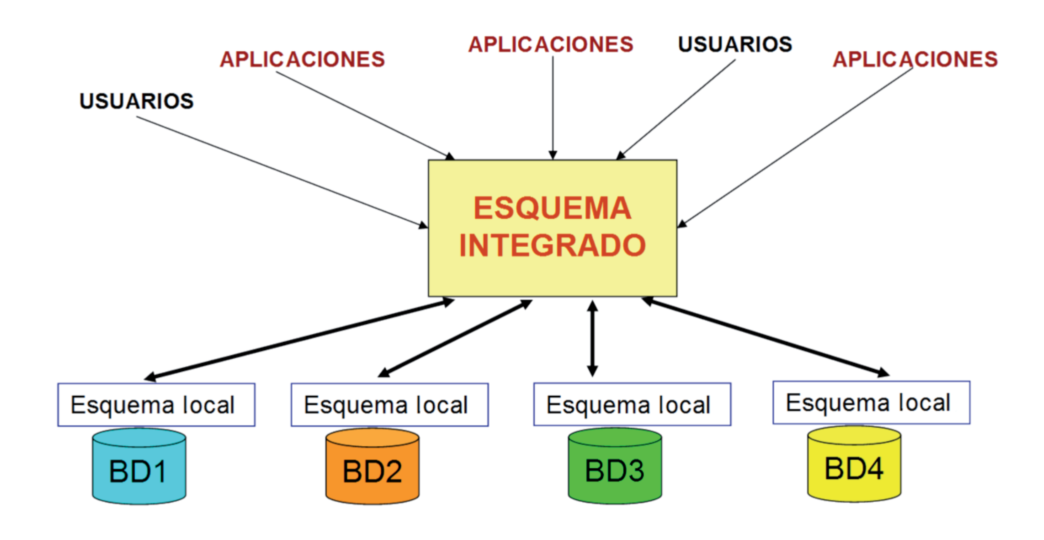


Figura 3.5. SBDD totalmente integrado

4.2. Heterogeneidad de datos y sistemas

La *heterogeneidad* de los SBDD se debe a que los nodos que lo componen pueden utilizar diferentes sistemas hardware, protocolos de red, lenguajes de consulta, protocolos de control de transacciones, e incluso diferentes modelos de datos como puede ser el relacional y el orientado a objetos. Un SGBDD puede soportar diferentes niveles de heterogeneidad haciendo que todos los problemas que eso conlleva sean transparentes para los usuarios. Muchas veces la heterogeneidad de los SBDD viene definida por los requisitos de las aplicaciones que deben soportar ya que pueden requerir manejar datos con formatos muy diferentes. Sin embargo, en otras muchas ocasiones el nivel de heterogeneidad del sistema es consecuencia de que el SBDD a construir resulta de la composición de unos sistemas no distribuidos que ya existían previamente y que habían sido desarrollados independientemente.

4.3. Distribución de los datos

Teniendo en cuenta la manera en que se distribuyen los datos por los nodos, los SBDD se pueden dividir en dos grupos. Por una parte, están los sistemas *peer-to-peer* o totalmente distribuidos, donde cada nodo de la red es un SGBD totalmente funcional, que además tiene la capacidad de poderse comunicar con otros nodos para ejecutar consultas y transacciones, y así compartir todos o algunos de sus datos. El caso opuesto es el de los *sistemas cliente/servidor*, que clasifican los nodos del SBDD en dos grupos separados: los nodos servidores y los nodos cliente. Los nodos cliente dependen completamente de los nodos servidores ya que les proporcionan el acceso a los datos compartidos, y viceversa, los nodos servidores solamente pueden ser accedidos a través de un nodo cliente. En estos sistemas, los nodos servidores gobiernan los accesos y gestionan los datos, mientras que los nodos cliente simplemente tienen capacidad de dar acceso para realizar consultas y ejecutar aplicaciones.

Dimensión	Autonomía	Heterogeneidad	Distribución	Las tres dimensiones
Nivel alto	Federados	Heterogéneos	Peer-to-Peer	Múltiples BD
Nivel bajo	Integrados	Homogéneos	Cliente/Servidor	BD centralizadas

Tabla 3.1. Principales tipos de SBDD

Como muestra la última columna de la tabla 3.1, las bases de datos centralizadas no presentan autonomía, ni heterogeneidad ni distribución de datos. En el otro extremo están los *sistemas de múltiples bases de datos* que presentan un alto nivel de autonomía, heterogeneidad y distribución. En realidad, los sistemas de múltiples bases de datos están formados por varias bases de datos que siendo totalmente independientes están dispuestas a proporcionan sus datos para que sean integrados a un esquema global. Por esta razón también se les denomina sistemas de integración de información. Al final del capítulo dedicaremos varias secciones a estudiar estos sistemas.

5. Diseño de bases de datos distribuidas

Al igual que cualquier otro sistema de bases de datos, diseñar un SBDD requiere analizar los requisitos de los usuarios y las aplicaciones para obtener un diseño conceptual, un esquema lógico y una organización física de los datos. Sin embargo, cuando los datos se distribuyen por una red de comunicaciones en varias bases de datos remotas, hay que tener en consideración otras cuestiones adicionales que tienen que ver con el nivel de compartición de los datos y los patrones de acceso más frecuentes. En esta sección primero se explica en qué consiste el diseño de la distribución de los datos, que es la etapa del diseño de bases de datos que trata de resolver los problemas que se relacionan con las cuestiones anteriores. Después se explican todas las etapas del proceso de diseño de SBDD.

5.1. Diseño de la distribución de los datos

El esquema global de un SBDD consiste en una colección de tablas relacionadas entre sí. Sin embargo, cada una de estas tablas no tiene necesariamente que materializarse en un único nodo del sistema. Normalmente, los contenidos de las tablas de una base de datos distribuida se dividen en fragmentos cuya localización se reparte por los distintos nodos. Cada fragmento se puede almacenar en uno o varios nodos produciendo lo que se llama réplica de datos. Durante el diseño de la base de datos distribuida se realiza el diseño de la distribución que consiste en decidir qué datos deben ser almacenados en cada uno de los nodos. El objetivo de esta tarea es conseguir que los datos estén localizados allí donde vayan a ser requeridos con mayor frecuencia. Toda la información concerniente a la distribución y fragmentación de datos se debe almacenar en el diccionario de datos, y su existencia debe ser transparente para los usuarios.

5.1.1. Fragmentación de datos

En un principio cuando se diseña la base de datos hay que decidir en qué nodo se ha de almacenar cada una de las tablas. Sin embargo, pueden existir tablas con muchas filas que deben ser accedidas desde nodos diferentes. En ese caso, puede ser conveniente dividir las filas de la tabla en grupos, de tal forma que cada grupo contiene aquellas filas de la tabla que son accedidas con más frecuencia desde cierto nodo. Por ejemplo, tal y como se muestra en la parte izquierda de la figura 3.6, si tenemos una empresa con tres departamentos distantes entre sí donde cada departamento cuenta con un nodo del SBDD, se puede dividir la tabla de empleados en tres fragmentos. Cada fragmento debe contener los empleados de uno de los departamentos y se almacenará en su correspondiente nodo. De esta manera, cada departamento tendrá los detalles de sus empleados en su nodo local. Este mecanismo se denomina fragmentación horizontal, donde un fragmento horizontal de una tabla es un subconjunto de las filas de dicha tabla que se almacena en un nodo diferente.

Alternativamente, un fragmento vertical de una tabla mantiene solo ciertas columnas de la tabla, es decir, la fragmentación vertical divide una tabla verticalmente por columnas. Por ejemplo, podemos dividir la tabla de empleados en dos fragmentos verticales, donde el primer fragmento incluya información personal (nombre, fecha de nacimiento, sexo) y el segundo información acerca del puesto de trabajo (salario, jefe inmediato, sede). Esta forma de fragmentación vertical no es totalmente correcta porque, si ambos fragmentos se almacenan por separado, no podremos reconstruir las filas originales. Para que sea posible reconstruir una tabla completa a partir de sus fragmentos, es necesario incluir la clave primaria de la tabla en todos los fragmentos verticales. En nuestro ejemplo de la parte derecha de la figura 3.6, tendríamos que añadir la clave primaria ident en los dos fragmentos.

Nodo de la sede de Barcelona

ident	nombre	fec_nac	sexo	jefe	sede	salario
13	Juan	12-02-1966	M	00	Barcelona	3000
34	Mario	03-05-1959	M	13	Barcelona	2000
23	Paula	30-11-1967	F	13	Barcelona	2000

Nodo de la sede de Madrid

ident	nombre	fec_nac	sexo	jefe	sede	3000
17	Jessica	18-02-1956	F	00	Madrid	2000
56	Ana	31-03-1958	F	17	Madrid	2000

Nodo de la sede de Valencia

ident	nombre	fec_nac	sexo	jefe	sede	salario
19	Salva	01-02-1956	M	00	Valencia	2500
32	María	03-06-1969	F	19	Valencia	2000
53	Pedro	30-01-1965	M	19	Valencia	2000

Nodo del departamento de Personal

ident	nombre	fec_nac	sexo
13	Juan	12-02-1966	M
34	Mario	03-05-1959	M
23	Paula	30-11-1967	F
17	Jessica	18-02-1956	F
56	Ana	31-03-1958	F
19	Salva	01-02-1956	M
32	María	3-06-1969	F
53	Pedro	30-01-1965	M

Nodo del departamento de Organización

ident	jefe	sede	salario
13	00	Barcelona	3000
34	13	Barcelona	2000
23	13	Barcelona	2000
17	00	Madrid	3000
56	17	Madrid	2000
19	00	Valencia	2500
32	19	Valencia	2000
53	19	Valencia	2000

Figura 3.6. Fragmentación horizontal y vertical de la tabla de empleados de una empresa

También es posible combinar ambos tipos de fragmentación para obtener una fragmentación mixta. Diseñar el esquema de fragmentación de una base de datos consiste en definir un conjunto de fragmentos que incluya todas las columnas y todas las filas de la base de datos, de manera que sea posible reconstruir la base de datos a partir de los fragmentos definidos. Un esquema de reparto describe la forma de distribuir los fragmentos definidos por los nodos de la base de datos.

Para saber qué nodos almacenan los datos que se requieren para poder procesar una consulta, el SGBDD utiliza la información sobre la distribución de datos que se almacena en el diccionario de datos. Para la fragmentación vertical, el diccionario guarda los nombres de las columnas de la tabla de cada fragmento. Para la fragmentación horizontal, se guarda la condición que satisfacen las filas de cada fragmento, por ejemplo, que el departamento sea el de Madrid, el de Valencia, etc.

5.1.2. Réplica de datos

Cuando un fragmento se almacena en más de un nodo se dice que está replicado. La réplica de datos resulta útil para mejorar la disponibilidad de los datos dado el gran número de usuarios que puede tener una base de datos distribuida. La desventaja de la réplica de datos es que puede reducir la rapidez de las operaciones de actualización. Una sola operación de actualización se deberá ejecutar con todas las copias del dato a fin de mantener la consistencia de la base de datos. El número de copias de cada fragmento puede ir desde uno hasta el número total de nodos en el sistema distribuido. Determinar en qué nodos se debe replicar cada fragmento de la base de datos se denomina diseñar un esquema de réplica.

La elección de los nodos y el grado de réplica dependen de los objetivos de rendimiento y disponibilidad del sistema y del tipo y la frecuencia de las transacciones que se ejecutan en cada nodo. Por ejemplo, si se trata de datos que requieren una alta disponibilidad en todos los nodos y las transacciones son en su mayoría de consulta por lo que son datos con pocas actualizaciones, entonces lo mejor es replicar el fragmento en todos los nodos. Por el contrario, si son datos que sufren muchas actualizaciones, puede ser conveniente limitar la réplica de datos. Diseñar el esquema de réplica y reparto de un SBDD es un problema de optimización bastante complejo que requiere analizar mucha información sobre el nivel de compartición de los datos y los patrones de acceso más frecuentes.

5.2. Etapas del diseño de SBDD

A la hora de abordar el diseño de un SBDD se puede optar entre dos estrategias: estrategias ascendentes y estrategias descendentes. Una estrategia ascendente está indicada para realizar el diseño del SBDD a partir de un número de bases de datos ya existentes con el fin de integrar sus datos. En este caso se dispone de los esquemas lógicos locales de las bases participantes por lo que la tarea de diseño se centra en crear un esquema lógico global donde se integran los datos proporcionados

por cada nodo. Con una estrategia descendente el proceso es similar al diseño de bases de datos centralizadas y produce un esquema lógico global. A continuación, en la fase de diseño de la distribución se define en qué nodos se debe almacenar cada dato, lo cual produce cambios al esquema lógico local de cada nodo. A continuación, se resumen brevemente las etapas de una estrategia genérica de diseño descendente.

Como se muestra en la figura 3.7, el proceso de diseño comienza con un análisis de los requisitos del SBDD. En esta primera etapa también se fijan otros objetivos del sistema como los niveles de rendimiento, seguridad, disponibilidad, flexibilidad y económicos. Como puede observarse, los resultados de este primer paso sirven de entrada para dos actividades que se realizan de forma paralela. El diseño de las vistas trata de definir los requisitos de datos de cada grupo de usuarios y aplicaciones, y por otro lado, el diseño conceptual se encarga de examinar la organización para determinar sus entidades, propiedades, relaciones y funcionalidades. Existe un vínculo entre el diseño de las vistas y el diseño conceptual, ya que el diseño conceptual debe corresponder con las vistas del usuario. Este aspecto es de vital importancia ya que el modelo conceptual debería soportar no solo las aplicaciones existentes, sino que debería estar preparado para futuras aplicaciones.

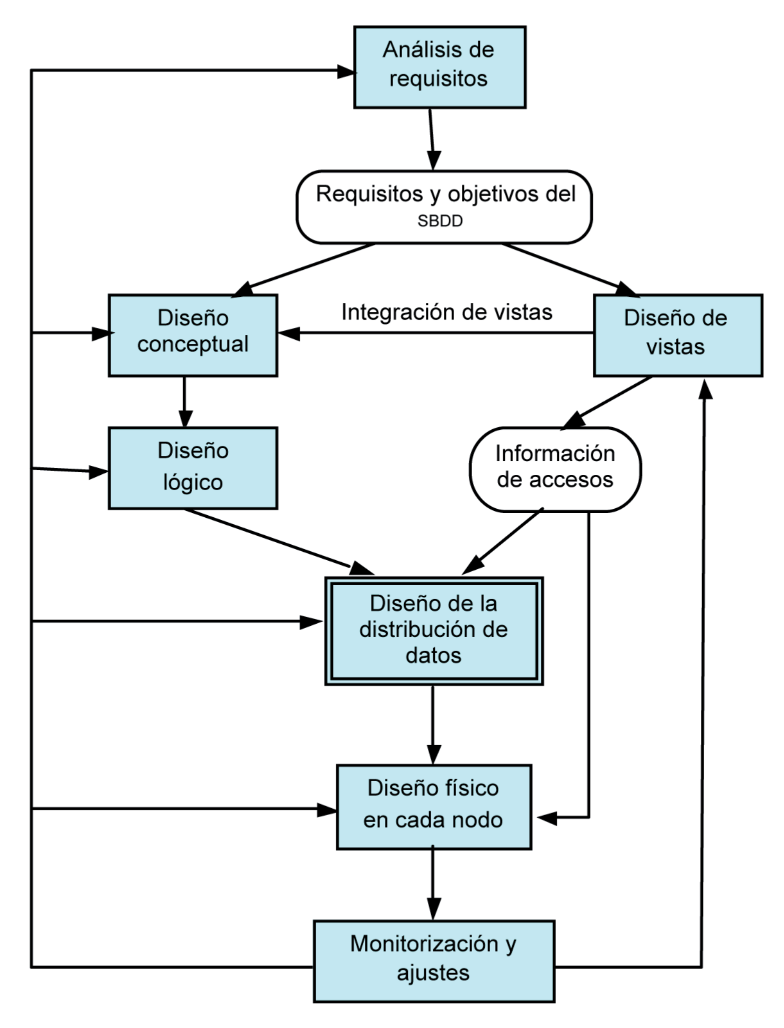


Figura 3.7. Etapas del diseño descendente de SBDD

En la etapa de diseño de vistas de usuario también se recopilan datos estadísticos y estimaciones sobre la frecuencia de acceso de las aplicaciones a las distintas tablas de las que hace uso, e incluso podrían anotarse las columnas a las que se accede. Toda esta información junto con el esquema lógico global creado a partir del diseño conceptual anterior, sirve de entrada al diseño de la distribución de datos.

El principal objetivo de la etapa del diseño de la distribución de datos consiste en diseñar los esquemas lógicos locales de todos los nodos del sistema distribuido. Tal y como se ha explicado en la sección 4.1, el proceso del diseño de la distribución consiste en situar los datos en aquellos nodos donde se vayan a necesitar más, y consta de las tres actividades explicadas anteriormente: el diseño de los esquemas de fragmentación, réplica y reparto de los datos. El último paso del diseño de un SBDD es el diseño físico, el cual utiliza la información de los accesos para implementar los esquemas lógicos locales sobre los dispositivos de almacenamiento físico disponibles en cada nodo. Por último, hay que tener en cuenta que la actividad de desarrollo y diseño de bases de datos es un tipo de proceso que necesita una monitorización y un ajuste periódicos, para que si se llegan a producir desviaciones con respecto a los objetivos iniciales, se pueda volver a alguna de las fases anteriores.

6. Procesamiento de consultas en bases de datos distribuidas

El procesamiento de consultas en un SBDD es más complicado que en una base de datos centralizada porque hay que tener en cuenta factores adicionales debido a que los datos se encuentran distribuidos por varios nodos remotos. Se deben transferir los datos que se necesitan para procesar la consulta, así como los datos del resultado final que deberán transferirse al nodo donde se solicitó la consulta. Los procesadores de consultas de los SGBDD tienen el objetivo de reducir la cantidad de datos a transferir por la red, siendo este el criterio más importante que tienen en cuenta para diseñar la estrategia de ejecución de una consulta distribuida.

Un SGBDD que soporte transparencia de localización de los datos permitirá al usuario especificar sus consultas utilizando el esquema lógico global y de la misma manera que si se tratara de un sistema centralizado. Para conocer en qué nodos se encuentra la información que se necesita para procesar la consulta, el procesador utiliza la información almacenada en el diccionario de datos. Una vez ya se conoce la localización de los datos, la consulta se descompone en varias subconsultas que se ejecutan en nodos individuales. Además, debe elaborarse una estrategia para combinar los resultados de las subconsultas y formar el resultado final. El procesador estimará el coste de ejecutar la consulta distribuida con varias estrategias alternativas y escogerá la mejor. Para poder estimar los tiempos de ejecución de cada estrategia, el SGBDD dispone en el diccionario de datos de información estadística acerca de los datos almacenados y la velocidad de la red de comunicaciones.

Con el fin de justificar la necesidad de encontrar una buena estrategia de ejecución, en esta sección veremos un ejemplo de procesamiento de consulta distribuida y estimaremos el coste de varias estrategias de ejecución. Después estudiaremos la técnica de los semi-*joins* que permite procesar consultas al mismo tiempo que se reduce la cantidad de datos a transferir. Para finalizar, enumeraremos las etapas de todo el procesamiento de consultas en SBDD.

6.1. Ejemplo de procesamiento

Supongamos que tenemos la siguiente base de datos distribuida de la forma indicada:

HOTELES (código, nombre, ciudad, país, categoría) con 500 filas en el nodo de Barcelona.
CLIENTES (dni, nombre, ciudad, país) con 50.000 filas en el nodo de Alicante.
RESERVAS (dni, código, precio, periodo) con 1.000.000 filas en el nodo de Alicante.

Se quiere procesar una consulta que seleccione los clientes de Francia que han hecho alguna reserva en un hotel de categoría superior. Para procesar esta consulta es necesario disponer de los datos de las tablas HOTELES, CLIENTES y RESERVAS para seleccionar las filas que se refieren a alguna reserva hecha por un cliente de Francia en un hotel de categoría superior.

Con los datos estadísticos de que se dispone en el diccionario de datos, el procesador de consultas del SGBDD es capaz de estimar que hay 20 hoteles de categoría superior y 150.000 reservas hechas por clientes de Francia. Supóngase también que la red tiene una capacidad de transmisión de 10.000 bits/s, el tamaño medio de las filas almacenadas es de 100 bits y que el tiempo de procesamiento de las operaciones de la consulta es despreciable en comparación con el tiempo de comunicación. Con estos datos, vamos a calcular de manera aproximada el tiempo que se tardarían en enviar los datos necesarios para ejecutar la consulta. Considerando constante el tiempo de envío de resultados al nodo donde se originó la consulta, se puede estimar el tiempo de comunicación de cuatro estrategias diferentes.

Estrategia 1. Llevar la tabla HOTELES al nodo de Alicante y procesar allí la consulta.

tiempo = 500*100/10.000 = 5 s

Estrategia 2. Llevar las tablas CLIENTES y RESERVAS al nodo de Barcelona y procesar allí la consulta:

tiempo = (50.000 + 1.000.000)*100/10.000 = 10.500 s = 3 h aprox.

Estrategia 3. Combinar las tablas CLIENTES y RESERVAS en el nodo de Alicante para seleccionar las filas del resultado que se corresponden con reservas hechas por clientes de Francia, y transmitir los datos seleccionados al nodo de Barcelona donde se completa el proceso.

tiempo = 150.000*(100+100)/10.000 = 3.000 s = 50 min

Estrategia 4. Seleccionar en el nodo de Barcelona las filas de la tabla de HOTELES que son de categoría superior y llevar el resultado al nodo de Alicante donde se completa el proceso.

tiempo = 20*100/10.000 = 0,2 s

Como puede observarse, de unas estrategias a otras el tiempo de comunicación puede variar mucho. Por lo tanto es crucial que el SGBDD pueda almacenar suficientes datos estadísticos para realizar una buena planificación de la estrategia a seguir antes de ejecutar cada consulta en un tiempo razonable.

6.2. Semi-*joins*

Supongamos que se necesita hacer el *join* de las tablas R y S localizadas en dos nodos remotos. En lugar de reunir ambas tablas en un solo nodo, la operación se ejecuta por medio de dos operaciones de semi-*join* de tal manera que la cantidad de datos a transmitir se reduce significativamente.

La estrategia consiste en enviar las columnas de la tabla R que intervienen en la condición del *join* al nodo donde está la tabla S. Una vez allí, se hace el *join* de dicha proyección con la tabla S, reduciéndose su tamaño al eliminar las filas que no cumplen la condición del *join*. Después, los atributos del *join* junto con los atributos de S que se requieran en el resultado final son proyectados y enviados al nodo de la tabla R donde se completa el *join* de las dos tablas. De esta forma, solo hay que transferir las columnas del *join* en una dirección y una selección de las filas y columnas de la tabla S en la otra dirección. En la mayoría de los casos, esta estrategia requiere mucho menos tiempo de transferencia y ejecución que enviar una tabla completa de un nodo a otro.

Veamos un ejemplo de cómo aplicar esta estrategia. Supongamos que las tablas PUERTOS y COMPAÑIAS se almacenan en nodos diferentes y hay que procesar una consulta para seleccionar información sobre los puertos y las compañías que estén ubicados en la misma ciudad.

PUERTOS (nombre, localización, superficie, ciudad, . . .) en el Nodo A.
COMPAÑIAS (número, denominación, responsable, ciudad, . . .) en el Nodo B.

Utilizaremos los operadores del álgebra relacional siguientes:

- El operador $PROY_{col}$ para proyectar columnas.
- El operador SEL_{cond} para seleccionar las filas que cumplan una condición.
- El operador $\bowtie_{cond}$ para hacer *joins*.

La consulta concreta se expresa de la siguiente manera:

$PROY_{nombre, localizacion, denominacion, responsable, ciudad}(PUERTOS \bowtie_{PUERTOS.ciudad=COMPAÑIAS.ciudad} COMPAÑIAS)$

Como puede observarse, en el resultado hay columnas de las dos tablas, y la condición de *join* se comprueba sobre la columna ciudad de ambas tablas. Vamos a describir los pasos necesarios para ejecutar esta consulta por medio de dos operaciones de semi-*join*:

Paso 1. En el nodo B, proyectar el atributo del *join* (COMPAÑIAS.ciudad) y transferirlo al nodo A:

$F = PROY_{ciudad}(COMPAÑIAS)$

Paso 2. En el nodo A hacer una operación de semi-*join* para conocer los detalles de los puertos que están en la misma ciudad que alguna compañía, y enviar el resultado al nodo B:

$G = PROY_{nombre,\ localizacion,ciudad}(PUERTOS\ _{PUERTOS.ciudad=F.ciudad}\ (F))$

Paso 3. Finalmente, ejecutar en el nodo B la otra operación de semi-*join* que sirve para conocer todos los detalles de las compañías y los puertos que están en la misma ciudad:

$PROY_{nombre,\ localizacion,denominacion,responsable,ciudad}(G\ _{G.ciudad=COMPAÑIAS.ciudad}\ COMPAÑIAS)$

Los pasos 2 y 3 son dos operaciones de semi-*join*. El paso 2 realiza el *join* en un sentido haciendo que la tabla PUERTOS se reduzca a aquellas filas que están en la misma ciudad que alguna compañía. Posteriormente en el paso 3 se realiza el *join* en el otro sentido y se obtienen los detalles de las compañías que se ubican en la misma ciudad que algún puerto. De esta manera, con dos dos operaciones de semi-*join*, se calculan las filas que cumplen la condición de *join*, es decir los detalles de cada compañía y cada puerto que están en la misma ciudad.

Cuando en el resultado del *join* solo deban aparecer columnas de una de las tablas, entonces el *join* solo tendrá que ejecutarse en un sentido (pasos 1 y 2 del algoritmo). Por ejemplo, considerar la consulta siguiente:

$PROY_{nombre,\ localizacion,\ ciudad}(PUERTOS\ _{PUERTOS.ciudad=COMPAÑIAS.ciudad}\ COMPAÑIAS)$

Como puede observarse, en el resultado solamente hay columnas de la tabla PUERTOS, y la condición de *join* se comprueba sobre la columna ciudad de ambas tablas. Por eso, debemos comenzar proyectando los atributos del *join* en la tabla COMPAÑIAS:

Paso 1. En el nodo B, proyectar el atributo del *join* (COMPAÑIAS.ciudad) y transferirlo al nodo A:

$F = PROY_{ciudad}(COMPAÑIAS)$

Paso 2. En el nodo A hacer una operación de semi-*join* para conocer los detalles de los puertos que están en la misma ciudad que alguna compañía:

$PROY_{nombre,\ localizacion,ciudad}(PUERTOS\ _{PUERTOS.ciudad=F.ciudad}\ (F))$

En este caso, la consulta puede ejecutarse mediante la transmisión de una cantidad reducida de datos del nodo B al nodo A, y una sencilla operación de *join* en el nodo A.

6.3. Pasos del procesamiento de consultas distribuidas

Después de que un usuario formule una consulta sobre el esquema de datos global de un SBDD, el principal objetivo del procesador de consultas es diseñar una estrategia de ejecución que sea eficiente en el sentido de que minimice el tiempo de comunicación de los nodos. Tras decidir qué nodos deben involucrarse en la ejecución de la consulta, esta es transformada en un grupo de subconsultas que se ejecutan en dichos nodos y que producen unos resultados intermedios que sirven al procesador de consultas para construir el resultado final.

Más concretamente, para procesar una consulta en un SBDD, lo primero que debe hacer el SGBDD es localizar los datos que se requieren para evaluar la consulta. Para realizar esta operación todos los nodos de la base de datos disponen de acceso al diccionario de datos donde consultan los esquemas de fragmentación, réplica y reparto de los datos. Para elaborar una estrategia eficiente, los nodos también disponen en el diccionario de datos de información y datos estadísticos de los diferentes parámetros de funcionamiento de cada base de datos y de la red de comunicaciones (rendimientos de los dispositivos, volumen de los datos, velocidad de transmisión, etc.). Como ya hemos visto, se deben considerar todas las alternativas de ejecución de la consulta y, tras escoger la mejor, dividir la consulta global en un conjunto de subconsultas individuales a las bases de datos de los nodos involucrados. Dado que el objetivo principal es reducir el envío de datos por la red, la estrategia final reemplazará las operaciones de *join* con operaciones de semi-*join* siempre que sea rentable.

Después de elaborar una estrategia global que minimice el tiempo de comunicación entre nodos, se generan las subconsultas y se envían a los nodos donde deben ser ejecutadas (ver figura 3.8). Debido a que el SGBD que tiene cada nodo puede ser diferente, las consultas necesitan ser traducidas al lenguaje adecuado para el nodo donde van a ser ejecutadas. Entonces, cada subconsulta es optimizada con respecto a su esquema local y ejecutada individualmente en su nodo. Para ello se utilizan las mismas técnicas que en las bases de datos centralizadas (optimización sintáctica y estimación de costes de ejecución). Finalmente, los resultados parciales son enviados al nodo origen donde son combinados para obtener el resultado final.

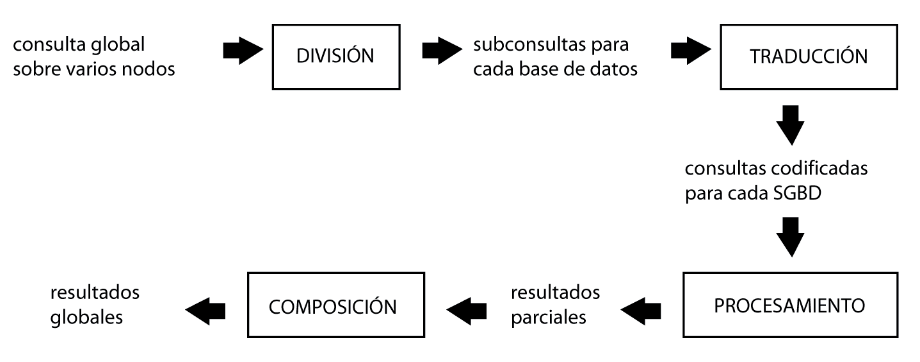


Figura 3.8. Etapas del procesamiento de consultas distribuidas

7. Propagación de actualizaciones

Para poder soportar la réplica de datos, en las bases de datos distribuidas las operaciones de actualización deben ser siempre propagadas a todas las copias existentes. Esta labor garantiza la consistencia de los datos almacenados y, aunque es transparente para el usuario, puede requerir bastante tiempo de ejecución.

A medida que una transacción va actualizando todas las copias de un dato, el mayor problema surge cuando alguno de los nodos que contienen una copia a actualizar se encuentra desconectado por algún fallo en la red de comunicación o en el nodo. Una manera de resolver este problema consiste en actualizar el dato en todas las copias disponibles, y a continuación introducir la actualización no realizada en una lista de actualizaciones pendientes. Cuando un nodo sea reconectado a la red, y antes de empezar a trabajar con normalidad, el procedimiento de reinicio tendrá que examinar esta lista para ver si el nodo tiene alguna actualización pendiente.

Otra posibilidad que permite a la transacción continuar ejecutándose sin tener que esperar a modificar todas las copias de un dato, y que por lo tanto ofrece mayor velocidad en las operaciones de actualización, consiste en designar una de las copias de cada dato como copia primaria. Las operaciones de actualización comienzan siempre por la copia primaria y si se hace con éxito, la transacción continua ejecutándose mientras que el nodo que contiene la copia primaria se encarga de propagar la actualización a todos los nodos con una copia del dato. Para repartir la tarea entre todos los nodos, las copias primarias de diferentes datos están en diferentes nodos. Sin embargo, con este método surge un problema cuando el nodo con la copia primaria se encuentra inaccesible. Para evitarlo, se puede hacer que una modificación se dirija a cualquier copia del dato, y que el nodo que la contiene la dirija al nodo con la copia primaria si está accesible o que designe a otro nodo con este papel.

7.1. *Snapshots* o instantáneas

Un SBDD que emplea esta técnica, en lugar de soportar varias réplicas de sus datos, mantiene *snapshots* o instantáneas. Una instantánea es el resultado de ejecutar una vista previamente definida sobre una o varias tablas de la base de datos. Dicho resultado es almacenado en algún lugar donde puede ser consultado directamente sin necesidad de acceder a las tablas originales. Periódicamente, las instantáneas son refrescadas, es decir, la vista es ejecutada y el nuevo resultado sustituye al antiguo. A continuación se da un ejemplo de definición en SQL de una instantánea que se refresca semanalmente.

```
CREATE SNAPSHOT hoteles_de_madrid
REFRESH FAST
      START WITH sysdate
      NEXT sysdate + 7
AS SELECT codigo, nombre, categoria FROM hoteles WHERE ciudad = 'Madrid';
```

Con esta técnica, en el SBDD solamente habrá una copia maestra de cada dato, y además se crearán las instantáneas necesarias para satisfacer la mayor parte de las consultas que se realicen. Las instantáneas son de solo lectura, por lo que las operaciones de lectura normalmente se dirigen hacia ellas. Lógicamente, para datos que deban soportar muchas operaciones de lectura, se pueden definir instantáneas que se almacenen replicadas en más de un nodo. Las operaciones de actualización deben dirigirse a las copias maestras y cuando se ejecuta una operación de refresco se propaga a las instantáneas. Las aplicaciones y los usuarios también pueden ejecutar operaciones de refresco cuando lo consideren necesario.

Aunque con esta técnica también existe el problema de la disponibilidad del nodo con la copia maestra, tiene la ventaja de que es muy simple y hace que las actualizaciones se ejecuten muy rápidamente. Hay que tener en cuenta que las instantáneas solamente se pueden utilizar para aplicaciones que no requieran tener los datos completamente actualizados, y que las operaciones de refresco son costosas, por lo que deben realizarse en momentos en los que el sistema tenga poca carga de trabajo. Por otra parte, sobre las instantáneas no pueden definirse índices, con lo que su acceso puede ser más lento de lo deseable.

8. Integración de información distribuida

Durante las últimas décadas y desde todos los ámbitos se ha ido produciendo una ingente cantidad de información y datos electrónicos. Estas fuentes de datos presentan características heterogéneas ya que cada una tiene un origen diferente, se representa con su propio formato, y proporciona sus particulares mecanismos de acceso. Desde hace unos años, gracias a la expansión de Internet, estos datos pueden ser fácilmente transmitidos, lo que ha permitido que sistemas de información más o menos remotos interaccionen entre sí para intercambiar información distribuida. Por eso, hoy en día disponemos de un amplio abanico de nuevas tecnologías y arquitecturas que facilitan la implementación de sistemas en los que se integra información heterogénea originariamente distribuida por varias fuentes.

Tal y como ilustra la figura 3.9, los datos son heterogéneos porque tienen propiedades muy diferentes. La heterogeneidad puede deberse a que los sistemas de acceso a los datos son diferentes (SGBD, lenguajes de consulta...). Si los modelos de representación de los datos (relacional, orientado a objetos, XML, etc.) son también diferentes el nivel de heterogeneidad es aún mayor. Otro tipo de heterogeneidad se da a nivel de los datos y ocurre cuando el significado, el formato de representación o la interpretación de los datos varia de unas fuentes a otras. Por ejemplo, dependiendo de su origen, el atributo precio se puede llamar coste, puede incluir impuestos, puede ponerse con decimales, puede ser total o parcial, etc.

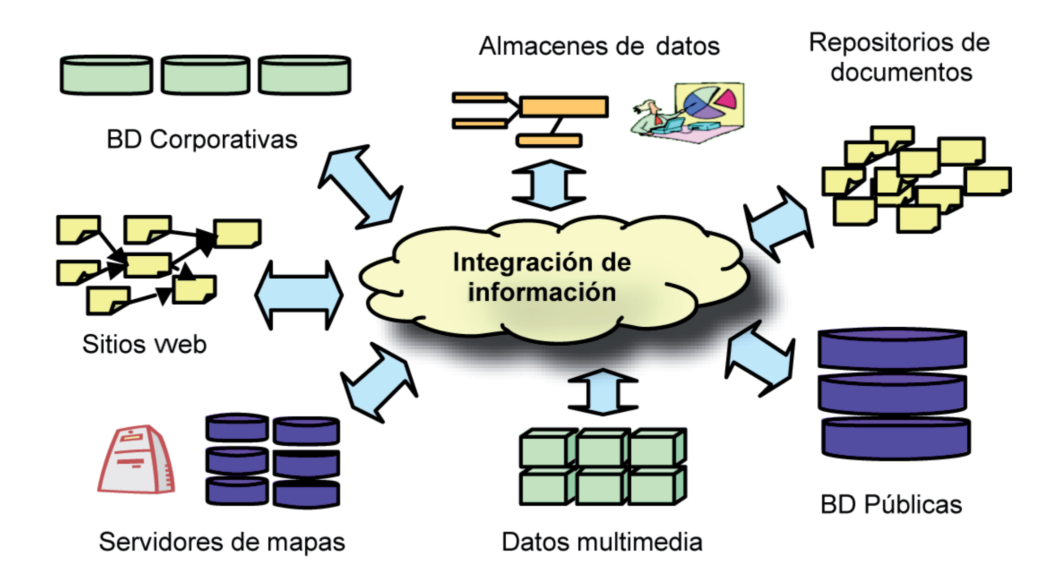


Figura 3.9. Integración de fuentes de datos heterogéneos

Cada día hay más aplicaciones y sistemas de información que requieren integrar información distribuida y heterogénea, algunos ejemplos se enumeran a continuación:

- Sistemas de información empresarial integrada: sistemas ERP, CRM y *workflow*.
- Aplicaciones de análisis de información empresarial: sistemas de apoyo a la toma de decisiones, herramientas OLAP, minería de datos...
- Plataformas integradas para la gestión de información médica y hospitalaria.
- Museos virtuales y bibliotecas digitales.
- Herramientas de periodismo electrónico.
- Aplicaciones de comercio electrónico.
- Bioinformática y biotecnología: explotación de datos provenientes de investigaciones y estudios biológicos y biomédicos.

Esta sección se dedica a explicar brevemente en qué consiste el proceso de integrar información, la reconciliación de los datos y las principales arquitecturas que se han diseñado para facilitar el desarrollo de todas estas aplicaciones y sistemas de información.

8.1. El proceso de integrar información distribuida

Todas las aplicaciones que hemos enumerado anteriormente tienen en común que requieren integrar información heterogénea y distribuida. Sin embargo, cada una de ellas presenta sus propias características y requerimientos, pudiendo éstos llegar a ser muy complejos. Aunque la tecnología actual permite resolver la mayor parte de los problemas que conlleva la integración de información, no existe ningún método ni procedimiento que pueda ser utilizado para decidir cómo desarrollar una solución completa a un problema concreto de integración de información.

Cada problema es diferente y, después de estudiar sus peculiaridades, habrá que escoger, de entre las muchas opciones que hay disponibles, la tecnología adecuada que facilite el desarrollo de una solución. No obstante, a continuación se presenta un procedimiento genérico que puede ayudar a analizar y resolver una problema de integración de información. Los pasos son los cuatro siguientes:

1. **Identificación de las fuentes de datos.** Analizar las aplicaciones para determinar qué datos se requiere integrar y dónde encontrarlos. A veces, será necesario seleccionar los mejores sitios donde obtener los datos que se requieren. En otros casos, los datos a integrar y su origen vienen establecidos por la aplicación a resolver.

2. **Análisis de las fuentes de datos.** El análisis consistirá en estudiar las propiedades de los datos a integrar (claves, relaciones, dominios, reglas de integridad...). Además, hay que generar los metadatos que se van a requerir en todo el proceso. Los metadatos nos dan información sobre los datos a integrar, y son necesarios para que las aplicaciones y los usuarios conozcan sus propiedades y puedan manipularlos y entenderlos mejor.

3. **Reconciliación y normalización de los datos.** La reconciliación de datos consiste en determinar la mejor manera de representar cada dato (dominios, estructuras de datos, formatos, etc.), ocultando las diferencias que ocurren especialmente para aquellos datos que vengan de varias fuentes diferentes. El objetivo es generar una forma normalizada que permita acceder a los datos de una manera homogénea. Se deberán implementar unos mecanismos para recuperar los datos con su forma original y transformarlos a la forma normalizada que se haya establecido. Para ello, se deben resolver complicados problemas de reconciliación como son: identificación de entidades, redundancias y contradicciones, ausencia y pérdida de información, significado diferente de los datos, etc.

4. **Especificación, diseño de la integración e implementación del sistema.** En esta fase final se requiere ejecutar las siguientes tareas:

   - Diseñar un modelo de datos integrado. Se deberá crear un esquema de datos global donde se integren todos los datos. Este esquema será el punto de acceso de los usuarios y las aplicaciones a los datos integrados.
   - Diseñar una arquitectura para la implementación del sistema de integración. Hay muchas posibilidades como un almacén de datos materializados, un esquema de integración virtual, una integración basada en servicios web...
   - Escoger la tecnología que se va a emplear para implementar los elementos del sistema: SGBD, protocolos de comunicación, plataformas tecnológicas, infraestructuras de desarrollo...
   - Ejecutar y desarrollar el software necesario para crear el sistema integrado: implementar el esquema lógico de integración, materializar los datos integrados, crear los índices, implementar los interfaces...

8.2. Reconciliación de datos para su integración

La integración de datos heterogéneos provenientes de fuentes distribuidas es una actividad compleja que implica la reconciliación de los datos a varios niveles. Reconciliar los datos significa encontrar una forma común para representar los datos integrados ocultando sus diferencias originales. En concreto, se debe definir sistemas de reconciliación en los siguientes tres niveles:

- **Modelos de datos.** Las fuentes de datos a integrar pueden diferir en gran medida con respecto a las estructuras que utilizan para representar los datos: tablas de bases de datos relacionales, objetos de bases de datos orientadas a objetos, repositorios de imágenes, páginas web, etc. Para poder integrar datos representados con diferentes estructuras, es necesario reconciliar sus modelos de datos en un modelo común capaz de expresar las características de todos ellos. En los últimos años, se ha demostrado que el modelo de datos de XML es capaz de representar las características de cualquier otro modelo de datos de una manera muy natural. Es decir, en XML se pueden representar fácilmente tanto tablas, como objetos, como páginas web, así como definir un modelo de datos donde se integren todos ellos. Por esta razón, XML es también considerado un lenguaje de integración de datos.

- **Esquema de datos.** Una vez que han acordado un modelo de datos común, se deben reconciliar las diferentes representaciones que se hagan de la misma entidad o propiedad. Es frecuente que ocurran cosas como que dos fuentes de datos estén utilizando distintos nombres para representar al mismo concepto (por ejemplo, precio y coste), el mismo nombre para representar conceptos diferentes (por ejemplo, un título puede ser muchas cosas), o dos maneras diferentes de expresar la misma información (por ejemplo, fecha de nacimiento y edad). Además, las fuentes de datos pueden representar la misma información utilizando diferentes estructuras de datos (por ejemplo, listas y *arrays* para almacenar un conjunto de datos). A nivel de esquema de datos, la reconciliación de datos consiste en definir una única representación y estructura de datos para cada elemento de información a manejar.

- **Instancias de datos.** Cuando se integran datos provenientes de varias fuentes pueden surgir varias versiones de la misma información, y no es raro encontrar versiones contradictorias. Por ejemplo, encontrar dos edades diferentes para la misma persona. Este es un problema de reconciliación de datos muy difícil de resolver ya que es necesario determinar cuál de las diferentes versiones es la correcta. Si se dispone de metadatos acerca de los datos, este problema puede ser más fácil de resolver. Por ejemplo, si se conoce la fecha en que un dato ha sido actualizado, se sabrá que la copia más reciente es probablemente la versión válida.

8.3. Arquitecturas para integrar información distribuida

En esta sección se explican las principales opciones que hay a la hora de definir la arquitectura de una aplicación que integra información proveniente de varias fuentes distribuidas. La lista de arquitecturas que se enumeran a continuación sirve para entender las distintas posibilidades que hay para diseñar una aplicación que integra información distribuida por varias fuentes de datos y los diferentes niveles de integración que se pueden alcanzar con cada una de ellas. Sin embargo, es importante saber que esta lista no es exhaustiva porque, en realidad, la arquitectura de un sistema concreto deberá ser diseñada en base a las características de la aplicación y la tecnología adoptada. Presentamos a continuación cinco arquitecturas ordenadas de menor a mayor según el nivel de integración de datos que proporcionan.

1. **Interfaz de integración (figura 3.10).** El usuario o aplicación del sistema cuenta con una interfaz para acceder a los datos distribuidos que una vez recuperados son visualizados. En este caso, la integración de datos es mínima ya que solo se hace en la interfaz de visualización donde los datos recuperados son presentados sin normalizar y sin procesar. Por ejemplo, el motor de búsqueda Google puede ser considerado una interfaz de integración de información, ya que como respuesta a una consulta de usuario presenta en una página web una lista de sitios web que elabora a partir de un índice. Sin embargo, la integración real de la información devuelta debe ser hecha por el usuario después de procesarla por sí mismo. En otros casos la interfaz de integración puede ejecutar algunas tareas sencillas de integración y normalización de los datos originales. Por ejemplo, el Google Académico (http://scholar.google.es/) visualiza la lista de enlaces a las publicaciones de un autor junto con el número de citas que ha recibido. Sin embargo, esta aproximación es inviable para problemas de integración más complejos.

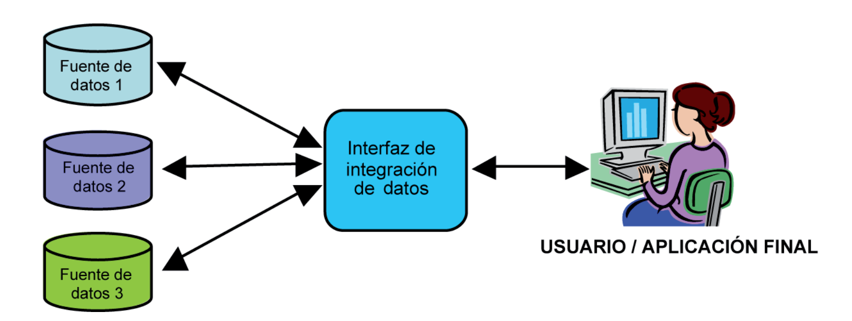


Figura 3.10. Interfaz de integración de datos

2. ***Middleware* de integración (figura 3.11).** Los usuarios y las aplicaciones del sistema disponen de diferentes herramientas *middleware* para ejecutar algunas de las funciones de integración de datos. El resto de funciones de integración las realizan ellos mismos con los datos que les proporcionan estas herramientas. Dependiendo de la tarea que realice, el *middleware* ocupará una posición u otra dentro de la arquitectura final. Por ejemplo, los recubridores se

encargan de proporcionar un acceso uniforme a cada fuente de datos, ocultando las diferencias en los mecanismos de acceso y los modelos de datos. Los mediadores se sitúan por encima de los recubridores ya que se encargan de acceder a las fuentes de datos y de realizar algunas tareas de integración y normalización de datos. Cuando se requiera, puede haber otros elementos *middleware* por encima de los mediadores, para encargarse de tareas de coordinación e intermediación. La arquitectura final puede tener varias capas de elementos *middleware* interactuando entre sí de acuerdo con la lógica del sistema a desarrollar.

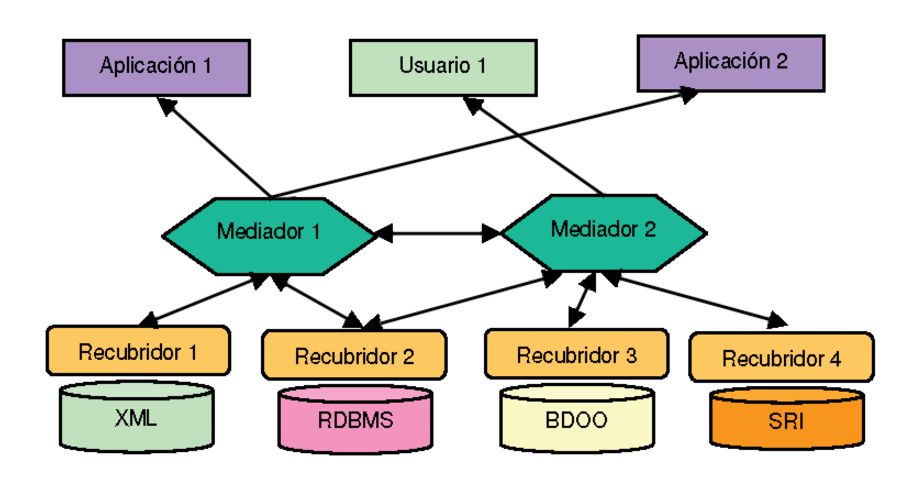


Figura 3.11. *Middleware* de integración de datos

3. **Esquema de integración virtual (figura 3.12).** Los usuarios y las aplicaciones acceden a los datos integrados a través de un esquema de datos global que realiza una integración virtual. Es decir, la integración es realizada por una capa intermedia en el momento en que los datos son solicitados. Sin embargo, los datos solamente se almacenan físicamente en su fuente de datos origen. El esquema de integración virtual realiza las tareas de integración de datos, y cuando necesita acceder a los datos en su origen, dispone de unos recubridores que transforman los datos originales al modelo de representación utilizado por el esquema integrado. Las consultas sobre el esquema integrado son traducidas automáticamente en subconsultas dirigidas a las fuentes de datos involucradas, y sus respuestas son integradas por la capa intermedia para generar los resultados de la consulta inicial.

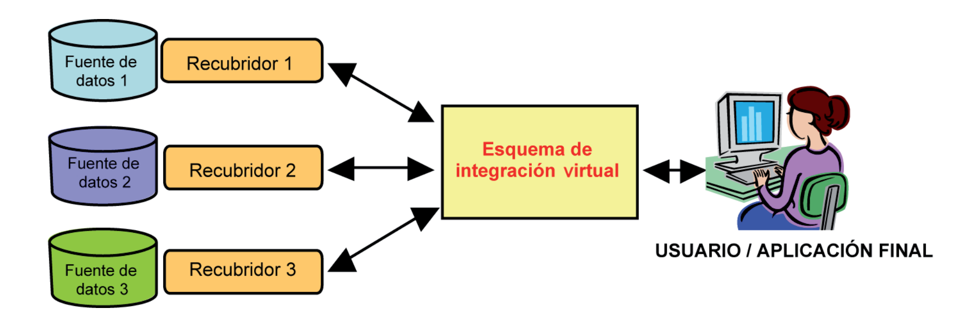


Figura 3.12. Esquema de integración virtual de datos

4. **Arquitectura de servicios web (figura 3.13).** Como antes, los usuarios y las aplicaciones acceden a los datos integrados a través de un esquema de datos global que realiza una integración virtual, y los datos solamente se almacenan físicamente en su origen. La principal diferencia de esta arquitectura con respecto a la anterior, es que el acceso a las fuentes de datos se realiza por medio de servicios web en lugar de a través de recubridores. Un *servicio web* es una pieza de software que puede ser ejecutada remotamente a través de Internet. Una de sus principales aplicaciones es la recuperación de datos de sus bases de datos locales para dar servicio a peticiones remotas. Los servicios web presentan la ventaja de mejorar la interoperatividad, ya que se basan en estándares abiertos que facilitan la comunicación y permiten que las aplicaciones y los propios servicios puedan evolucionar independientemente sin que afecten al resto del sistema.

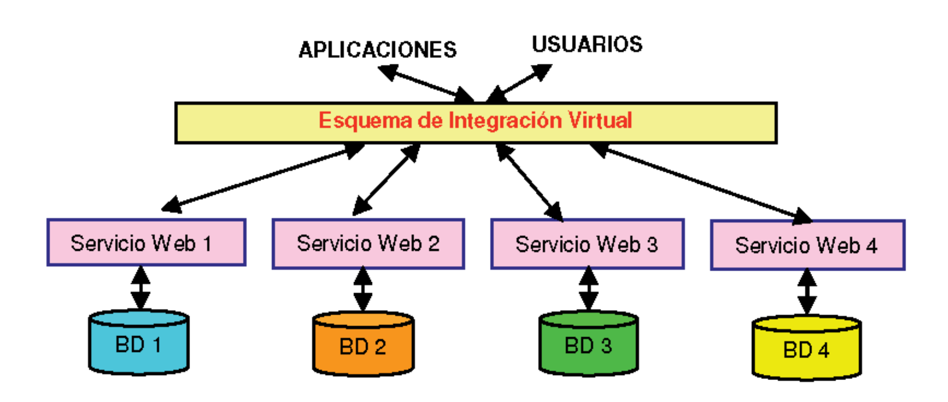


Figura 3.13. Arquitectura de servicios web

5. **Almacén de datos común (figura 3.14).** Las aplicaciones y los usuarios reciben datos ya integrados, a los que acceden a través de un esquema global que se materializa en un almacén de datos común. En este almacén se guardan los datos integrados, una vez trasladados desde su origen, y donde deben ser periódicamente refrescados. Para ello, estos sistemas cuentan con unos procesos denominados ETL, que se encargan de extraer los datos en su origen, transformarlos en un formato normalizado y cargarlos (*load*) en el almacén común (*Data Warehouse*). Los procesos ETL realizan todas las funciones necesarias para integrar los datos previamente a ser almacenados. Estos sistemas están orientados al análisis de información empresarial (*Business Intelligence*), por lo que soportan consultas complejas provenientes de usuarios especializados y de aplicaciones de *Reporting* y herramientas OLAP (*On-Line Analytical Processing*).

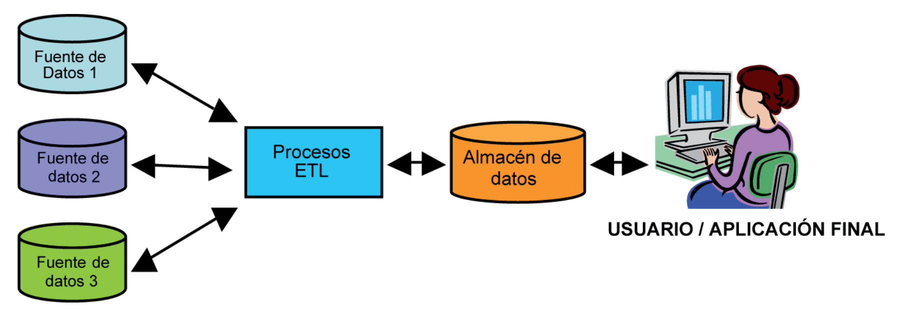


Figura 3.14. Almacén de datos común

Bibliografía

Bernstein, P. A., Haas, L. M. (2008): *Information integration in the enterprise,* Communications of ACM, ACM 51(9).

Bouguettaya, A., Benatallah, B., Elmagarmid, A. (1998): *Interconnecting Heterogeneous Information Systems*, Kluwer.

Elmasri, R., Navathe, S. (2011): *Fundamentals of Database Systems,* (6.ª edición), Addison-Wesley Longman.

Özsu M. T., Valduriez, P. (2011): *Principles of Distributed Databases,* (3.ª edición), Springer.

Pardede, E. (2009): *Open and Novel Issues in XML Database Applications: Future Directions and Advanced Technologies*. Information Science Reference.

Ejercicios

Ejercicio 1

Diseña una base de datos formada por al menos dos tablas diferentes distribuidas por al menos tres nodos separados. En base a las características de los datos y a las consultas previstas, realiza un esquema de fragmentación, réplica y reparto para dicha base de datos.

Ejercicio 2

Una empresa de turismo con sedes en Madrid y Barcelona decide promocionarse lanzando una campaña de descuentos especiales dirigidos a un determinado grupo de clientes. Este gasto se vería compensado realizando ciertos ajustes a los descuentos de otro tipo de clientes. Con este fin, tiene previsto el lanzamiento de las siguientes transacciones:

T1: Seleccionar aquellos clientes que teniendo una antigüedad superior a cinco años, no efectúan ningún viaje este verano.
T2: Seleccionar aquellos clientes que van a participar en más de un viaje durante el presente año.

Supongamos que dicha empresa trabaja en un entorno distribuido con nodos en Madrid y Barcelona cuya información está contenida en las siguientes tablas:

```
CLIENTES(dni, nombre, fecha_inscripción, descuento)
VIAJES(código, destino, duración)
RESERVAS(dni, código, fecha_salida)
```

Se pide plantear todas las estrategias de evaluación para las consultas T1 y T2 que permitan estimar los siguientes datos estadísticos almacenados en el diccionario de datos del sistema. Junto con cada estrategia hay que calcular su tiempo total de comunicación.

- CLIENTES tiene 10.000 filas en Barcelona.
- VIAJES tiene 1.000 filas en Madrid.
- RESERVAS tiene 1.000.000 filas en Madrid.
- Hay 1.000 clientes cuya antigüedad es mayor de 5 años.
- Efectúan algún viaje este verano 6.000 clientes.
- Efectúan más de un viaje este año 3.000 clientes.
- Velocidad de transmisión de 1.000 bits/s.
- Tamaño medio de fila es de 100 bits.

Ejercicio 3

En una compañía de seguros se dispone de una base de datos distribuida por tres nodos de la siguiente manera:

AUTOMÓVIL(matricula, dni_cond, marca, modelo, núm_póliza) en Valencia.
CONDUCTOR(dni, nombre, dirección, edad, sexo) en Barcelona.
SINIESTRO(matricula, dni, lugar, fecha, hora) en Sevilla.

Se desea ejecutar una consulta para seleccionar los conductores varones que teniendo más de tres coches, no hayan tenido ningún siniestro. Para ello se dispone de los siguientes parámetros:

- La tabla AUTOMÓVIL tiene 300.000 filas, CONDUCTOR tiene 150.000 filas y SINIESTRO tiene 400.000 filas.
- Hay 1.000 conductores varones con más de tres coches.
- Hay 30.000 conductores varones que no han tenido ningún siniestro.
- Hay 90.000 conductores varones.
- Hay 100.000 conductores que han tenido algún siniestro.
- La velocidad de transmisión de la red es de 1.000 bits/s.
- El tamaño medio de las filas almacenadas es de 100 bits.

Explicar una estrategia para evaluar esta consulta y estimar su tiempo de ejecución con los parámetros anteriores. No tener en cuenta ninguna estrategia que involucre enviar alguna tabla completa por la red.

Ejercicio 4

Se desea seleccionar de la siguiente base de datos distribuida el nombre, la edad, la marca y el modelo de los automóviles de cada conductor cuya profesión sea abogado:

AUTOMOVIL(matricula, dni_prop, marca, modelo, núm_póliza) almacenada en Valencia.
CONDUCTOR(dni, nombre, dirección, edad, sexo, profesión) almacenada en Barcelona.

Dada la siguiente expresión de esta consulta utilizando los operadores del álgebra relacional, explica detalladamente las operaciones que habría que ejecutar en cada nodo utilizando una estrategia con una o varias operaciones de semi-*join*:

$PROY_{nombre,\ edad,\ marca,\ modelo}$ ($SEL_{profesión='abogado'}$ ((AUTOMOVIL $\bowtie_{dni=dni_prop}$ CONDUCTOR)))

Ejercicio 5

Se desea seleccionar de la siguiente base de datos distribuida los detalles de los goles que ha metido cada jugador:

JUGADORES(dni, nombre, dirección, telefono, edad, prima, equipo) almacenada en Valencia.
GOLES(dni_jug, fecha, hora, tipo_gol, equipo_cont, campeonato, estadio) almacenada en Madrid.

Dada la siguiente expresión de esta consulta utilizando los operadores del álgebra relacional, explica detalladamente las operaciones que habría que ejecutar en cada nodo utilizando una estrategia con una o varias operaciones de semi-*join*:

$$\text{PROY}_{\text{nombre, prima, equipo, tipo_gol, equipo_cont}} \ (\text{JUGADORES} \bowtie_{\text{dni = dni_jug}} \text{GOLES})$$

Ejercicio 6

Se desea seleccionar de la siguiente base de datos distribuida el nombre y el departamento de los profesores que con menos de 50 años imparten asignaturas de más de 7 créditos:

PROFESORES(dni, nombre, dirección, teléfono, edad, departamento, grado) almacenada en París.
ASIGNATURAS(código, nombre, descripción, titulación, créditos, dni_prof) almacenada en Londres.

Especifica esta consulta utilizando el álgebra relacional y explica detalladamente las operaciones que habría que ejecutar en cada nodo de la base de datos anterior, para procesar esta consulta utilizando una estrategia con una o varias operaciones de semi-*join*.